ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΔΥΤΙΚΗΣ ΕΛΛΑΔΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

¨ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΜΗΧΑΝΙΣΜΟΥ ΑΣΦΑΛΕΙΑΣ ΜΕ ΤΗ ΧΡΗΣΗ ΔΙΑΤΑΞΕΩΝ ΥΛΙΚΟΥ ΓΙΑ ΔΙΚΤΥΑ ΤΕΤΑΡΤΗΣ ΓΕΝΙΑΣ (4G-LTE)¨

ΛΙΑΚΟΥ ΑΘΑΝΑΣΙΑ
Α.Μ. 1224

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ
ΚΙΤΣΟΣ ΠΑΡΑΣΚΕΥΑΣ

ΑΝΤΙΡΡΙΟ 2016

ΠΡΟΛΟΓΟΣ

Η παρούσα πτυχιακή εργασία με τίτλο ¨Σχεδιασμός και Υλοποίηση Μηχανισμού Ασφαλείας με τη χρήση Διατάξεων Υλικού για Δίκτυα Τέταρτης Γενιάς (4G-LTE)¨ υλοποιήθηκε στα πλαίσια της φοίτησης στο Τεχνολογικό Εκπαιδευτικό Ίδρυμα Δυτικής Ελλάδας στο Τμήμα Μηχανικών Πληροφορικής.

ΠΕΡΙΛΗΨΗ

Τα δίκτυα Τέταρτης Γενιάς (4G) αποτελούν την εξέλιξη των δικτύων Τρίτης Γενιάς και παρέχουν αναπτυγμένες υπηρεσίες για πρόσβαση σε ασύρματα δίκτυα. Μαζί με τις υπηρεσίες έχει βελτιστοποιηθεί και η ασφάλεια η οποία παρέχεται.

Σκοπός της παρούσας πτυχιακής εργασίας είναι η μελέτη, ο σχεδιασμός και η υλοποίηση ενός μηχανισμού ασφαλείας για τα δίκτυα Τέταρτης Γενιάς. Ο μηχανισμός ασφαλείας ονομάζεται MILENAGE και υλοποιήθηκε με βάση τον αλγόριθμο κρυπτογράφησης Rijndael. Αρχικά ορίζεται ο σχεδιασμός του μηχανισμού ασφαλείας. Η ανάπτυξη του κώδικα γίνεται με την γλώσσα περιγραφής υλικού VHDL, ο έλεγχος ορθής λειτουργίας γίνεται με το πρόγραμμα προσομοίωση ModelSim και τέλος η υλοποίηση του μηχανισμού σε υλικό γίνεται με την χρήση πολλαπλής χρήσης κυκλωμάτων, τα FPGAs.

Αρχικά γίνεται μία περιληπτική ιστορική αναφορά στα δίκτυα παλιότερων γενιών. Στο Κεφάλαιο 1 γίνεται μελέτη των δικτύων Τέταρτης γενιάς και της ασφάλειας που παρέχεται ενώ στο Κεφάλαιο 2 γίνεται αναφορά στους αλγόριθμους κρυπτογράφησης και ειδικότερα στον αλγόριθμο βάσης Rijndael και στις λειτουργίες του. Στο Κεφάλαιο 3 παρουσιάζονται και αναλύονται οι μηχανισμοί υλοποίησης ολοκληρωμένων κυκλωμάτων, ASICs και FPGAs και στο Κεφάλαιο 4 ορίζουμε τον σχεδιασμό του μηχανισμού ασφαλείας MILENAGE και τον κρυπτογραφικό αλγόριθμο Rijndael. Στην συνέχεια, στο Κεφάλαιο 5 γίνεται η ανάπτυξη του κώδικα με την χρήση της VHDL όπως επίσης και ο έλεγχος ορθής λειτουργίας του. Στο τελευταίο Κεφάλαιο, Κεφάλαιο 6, παρουσιάζεται το υλοποιημένο σύστημα με την χρήση FPGA και τα αποτελέσματα της σύνθεσης.

Λέξεις Κλειδιά: μηχανισμός ασφαλείας, MILENAGE, Rijndael, ασφάλεια, ασύρματα δίκτυα, 4G, δίκτυα τέταρτης γενιάς, σχεδιασμός, κρυπτογράφηση, VHDL, αλγόριθμος, ολοκληρωμένο κύκλωμα, FPGA.

ABSTRACT

4G networks are the development of 3G networks that provide advanced services to wireless networks. Along with the services, the security provided is optimized.

The purpose of this project is the design and implementation of a safety mechanism for 4G networks. The security mechanism called MILENAGE and the implementation based on the Rijndael encryption algorithm. Primary, the design of MILENAGE is defined. The development of code is made by the hardware description language, VHDL, the correct operation control is made with the simulation program ModelSim and finally the implementation of the mechanism in hardware is made by the use of FPGAs.

At the introduction, I make a brief historical reference to older generation networks. In Chapter 1, I will make a reference to 4G networks and to security that provided and in Chapter 2 I will refer cryptographic algorithms and in particular the base algorithm Rijndael and its operations. In Chapter 3, I will analyze the mechanisms of implementation of integrated circuits, ASICs and FPGAs and in Chapter 4 I will define the design of MILENAGE and Rijndael algorithm. Chapter 5 describes the development of the code using the hardware description language, VHDL. And finally, in Chapter 6, I shall present the implemented system using FPGA and the results of synthesis.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΕΙΣΑΓΩΓΗ

Εξέλιξη των Δικτύων Κινητής Τηλεφωνίας

Η ανάγκη των ανθρώπων για επικοινωνία ξεκινάει από την αρχή της ανθρώπινης ύπαρξης. Ως επικοινωνία νοείται η ανταλλαγή πληροφοριών μεταξύ 2 ή περισσότερων χρηστών. Η επικοινωνία ακολουθώντας την ανθρώπινη εξέλιξη, εξελίχθηκε και η ίδια λαμβάνοντας διαφορετικές μορφές. Για να φτάσουμε στις σημερινές μορφές επικοινωνίας (internet,κινητή τηλεφωνία) μεσολάβησε η ραγδαία ανάπτυξη της τεχνολογίας στον 20^{ο} αιώνα.

Με την εξέλιξη των Δικτύων Κινητής Τηλεφωνίας αναπτύχτηκαν γενιές δικτύων που παρουσιάζονται στην συνέχεια. Ως Γενιά – Generation (G) ορίζεται το σύνολο των ασύρματων τεχνολογιών που επιτρέπουν την επικοινωνία διαμέσου ενός δικτύου ασύρματης τεχνολογίας.

Συγκεκριμένα οι πρώτες προσπάθειες ανάπτυξης της κινητής τηλεφωνίας ξεκίνησαν με το τέλος του 2^{ου} Παγκοσμίου πολέμου. Η υλοποίηση αυτή της ιδέας επετεύχθη την δεκαετία του ‘80 με την Πρώτη Γενιά κινητής τηλεφωνίας (1G) η οποία χρησιμοποιούσε ασύρματο αναλογικό σύστημα. Χρησιμοποιήθηκε από πολλές χώρες της Αμερικής και της Ευρώπης με χαρακτηριστικά την χαμηλή ποιότητα μετάδοσης φωνής, τις μεγάλες και βαριές συσκευές επικοινωνίας, το ελάχιστο επίπεδο ασφάλειας και την έλλειψη υπηρεσιών. Τη δεκαετία του ‘90 ακολουθεί η Δεύτερη Γενιά κινητής τηλεφωνίας (2G) με την χρήση ψηφιακών συστημάτων. Ως ενδιάμεσο βήμα πριν την ανάπτυξη της Τρίτης Γενιάς (3G) μεσολαβεί η εξέλιξη του 2G σε 2,5G (GPRS τεχνολογία) και 2,7G (EDGE τεχνολογία), πρόκειται για μεταβατικές γενιές που προσφέρουν υψηλότερες ταχύτητες. Η εξέλιξη της Τρίτης Γενιάς (3G) ξεκινάει από τη δεκαετία του ‘90 και αναπτύσσεται πλήρως κατά την δεκαετία του 2000. Ο βασικός στόχος των δικτύων Τρίτης Γενιάς είναι να παρέχει στο χρήστη τη δυνατότητα να κάνει χρήση οποιασδήποτε υπηρεσίας, σε οποιοδήποτε μέρος, οποιαδήποτε στιγμή. Αυτό επιτυγχάνεται μέσω της υψηλής ταχύτητας μετάδοσης δεδομένων μοιρασμένων σε πακέτα (packet – switched).

Φτάνοντας στα τέλη της δεκαετίας του 2000 έχουμε την εμφάνιση της Τέταρτης Γενιάς δικτύων (4G), με την περαιτέρω ανάπτυξη της να λαμβάνει χώρα τις αρχές της δεκαετίας του 2010. Η Τέταρτη Γενιά (4G) αποτελεί ουσιαστικά μια εξέλιξη της προηγούμενης γενιάς. Με το δίκτυο 4G τα κινητά τηλέφωνα ουσιαστικά μετατρέπονται σε ηλεκτρονικούς υπολογιστές τα οποία θα μπορούν να υποστηρίζουν πολυμεσικές υπηρεσίες. Προσφέρεται δηλαδή στον χρήστης η δυνατότητα της χρήσης υπηρεσιών δεδομένων με πολύ μεγαλύτερες ταχύτητες, αποστέλλονται και λαμβάνονται αρχεία σε ελάχιστο χρόνο και εισάγεται η χρήση εφαρμογών πολυμέσων όπως η αναπαραγωγή High Definition βίντεο και το HD streaming.

Με την εξέλιξη των δικτύων αυξάνονται και οι ανάγκες ασφαλούς διακίνησης των δεδομένων στα δίκτυα καθώς οι χρήστες θα πρέπει να νοιώθουν πιο ασφαλείς κατά την επικοινωνίας τους.

Δίκτυα Δεύτερης Γενιάς

Τα δίκτυα Δεύτερης Γενιάς (2G) προσέφεραν πολλαπλές δυνατότητες σε σύγκριση με την Πρώτη Γενιά καθώς έκαναν χρήση ψηφιακού σήματος, αντί του αναλογικού. Ουσιαστικά έδωσε λύση στα προβλήματα συμβατότητας μεταξύ των διαφόρων συστημάτων που διέθετε η εκάστοτε χώρα. Διέθετε υψηλότερες ταχύτητες, ενώ προσέθεσε και άλλες υπηρεσίες όπως τα γραπτά μηνύματα και το e-mail. Επίσης παρείχε υψηλότερη ασφάλεια καθώς εισήγαγε την χρήση της ψηφιακής κρυπτογράφησης. Τα δίκτυα Δεύτερης Γενιάς χρησιμοποίησαν παρακάτω τέσσερα διαφορετικά πρότυπα.

- Global System for Mobile communications (GSM)
- Digital AMPS (D--AMPS)
- Code Division Multiple Access (CDMA) IS--95
- Personal Digital Cellular (PDC)

Από τα παραπάνω πρότυπα το πιο διαδεδομένο είναι το GSM (Παγκόσμιο Σύστημα Κινητών Επικοινωνιών). Η λειτουργία του GSM βασίζεται στην Κυψελοειδή Δομή Δικτύου. Η γεωγραφική κάλυψη μιας περιοχής γίνεται με το διαμοιρασμό της σε πολύ μικρές περιοχές που λέγονται κυψέλες. Κάθε κυψέλη διαθέτει έναν σταθμό βάσης και εφάπτεται με τις γειτονικές της έτσι ώστε να δημιουργείται μια ενιαία δομή. Για την κάλυψη μιας περιοχής επαναλαμβάνεται η δομή όσο συχνά χρειάζεται, κάνοντας επανειλημμένη χρήση των ίδιων συχνοτήτων, αυξάνοντας έτσι τη χωρητικότητα του δικτύου έχοντας όμως ως περιορισμό ότι η ισχύς της κάθε κυψέλης δεν πρέπει να υπερβαίνει τα όρια της ώστε να μην υπερχειλίζει άλλες κυψέλες της ίδιας δομής.

Η εξέλιξη του 2G στα στάδια του 2,5G και 2,7 G εισήγαγε νέες υπηρεσίες όπως το MMS και το WAP. Η διαφορά είναι ότι χρησιμοποιήθηκαν οι τεχνολογίες General Packet Radio Service(GPRS) και Enhanced Data rates for GSM Evolution (EDGE) οι οποίες ανέπτυξαν περαιτέρω την τεχνολογία GSM.

Δίκτυα Τρίτης Γενιάς

Με την ανάπτυξη των Δικτύων Τρίτης Γενιάς (3G) μπορούμε πλέον να έχουμε πρόσβαση στο παγκόσμιο ιστό από το κινητό, να επικοινωνούμε μέσω της υπηρεσίας Voice over Internet Protocol, να ‘ανεβάζουμε’ και να ‘κατεβάζουμε’ video και αρχεία μουσικής με μεγαλύτερες ταχύτητες, έως και 1.4 Mb/s καθώς επίσης παρέχονται αναβαθμισμένες υπηρεσίες. Επίσης τα 3G δίκτυα προσφέρουν υψηλότερη ασφάλεια σε σχέση με τις προηγούμενες γενιές.

Η πληθώρα υπηρεσιών που παρέχεται οφείλεται στην εξέλιξη των υπαρχόντων δικτύων 2G. Το κυριότερο πρότυπο που χρησιμοποιείται από τα 3G δίκτυα είναι το UMTS (Universal Mobile Telecommunications System) το οποίο αναπτύχθηκε από

την κοινοπραξία 3GPP (3rd Generation Partnership Project) στην οποία συμμετείχαν μερικοί από τους μεγαλύτερους οργανισμούς προτυποποίησης παγκοσμίως.

Αρχικός στόχος του 3GPP ήταν η δημιουργία τεχνικών προδιαγραφών και οδηγιών ο οποίοι θα είναι παγκοσμίως αποδεκτοί για συστήματα κινητής τηλεφωνίας Τρίτης Γενιάς, βασισμένο στους εξελιγμένους πυρήνες GSM δικτύων, καθώς και στις τεχνολογίες πρόσβασης που υποστηρίζουν.

Το κύριο πλεονέκτημα είναι ότι ο χρήστης έχει τη δυνατότητα ανταλλαγής πληροφοριών και χρήσης δεδομένων ακόμα και σε περιοχές όπου δεν υπάρχει κάλυψη δικτύου 3G.

ΚΕΦΑΛΑΙΟ 1
ΔΙΚΤΥΑ ΤΕΤΑΡΤΗΣ ΓΕΝΙΑΣ ΚΑΙ ΑΣΦΑΛΕΙΑ

1.1 Εισαγωγή

Οι ιδιαίτερα υψηλές ταχύτητες λήψης και αποστολής δεδομένων αποτελούν το κύριο χαρακτηριστικό των δικτύων τέταρτης γενιάς 4G προσφέροντας επίσης καλύτερη ποιότητα επικοινωνίας και καινοτομία στις υπηρεσίες.

Τα δίκτυα 4G είναι η διάδοχος γενιά των 3G δικτύων και σκοπός τους είναι να καλυφθούν τα κενά στις υπάρχουσες υπηρεσίες καθώς επίσης και να εισάγει νέες και πιο ασφαλείς. Τα δίκτυα Τέταρτης Γενιάς παρέχουν τις ίδιες υπηρεσίες με την 3G γενιά με κύρια διαφορά ότι ενώ στα 3G δίκτυα για να αποκτήσουμε πρόσβαση στο internet ή με υπολογιστή ή με κινητό πρέπει να είμαστε κοντά σε ένα wi-fi σημείο, δηλαδή με απόσταση το πολύ 100 μέτρων, τώρα έχουμε μεγαλύτερη ευρυζωνικότητα με την οποία μπορούμε να έχουμε πρόσβαση στο διαδίκτυο σε απόσταση περίπου 35 χιλιομέτρων από το Wi-MAX. Σκοπός της χρήσης της τεχνολογίας Wi-MAX είναι να γεφυρώσει το κενό ανάμεσα στις ασύρματες συνδέσεις και τις μεγάλες ταχύτητες των ενσύρματων συνδέσεων

Κλειδί της ανάπτυξης των 4G δικτύων είναι η ‘ενοποίηση’ των τερματικών, των εφαρμογών και των δικτύων ώστε να καλύπτονται οι ανάγκες των χρηστών. Συνεπώς κατά τον σχεδιασμό συμπεριλαμβάνονται συστήματα από διάφορα δίκτυα. Τα δίκτυα 4G βασίζονται δηλαδή στην δημιουργία και εγκατάσταση μιας υποδομής η οποία λειτουργεί ως συνδετικός κρίκος αλλά και ως κορμός του δικτύου είτε αφορούν δίκτυα κινητής τηλεφωνίας είτε ασύρματα δίκτυα δεδομένων.

<u>Στόχοι και Προκλήσεις Δικτύου Τέταρτης Γενιάς</u>

Ένα Δίκτυο Τέταρτης Γενιάς πρέπει να παραμένει σταθερό και να παρέχει υψηλό ρυθμό μετάδοσης δεδομένων. Ακόμα, οι αυξανόμενες απαιτήσεις των χρηστών δημιουργεί την ανάγκη της παραμετροποίησης και προσαρμοστικότητας στον καθένα. Ακόμα, οι υπηρεσίες θα πρέπει να είναι ενοποιημένες και να παρέχονται ταυτόχρονα από διαφορετικούς παρόχους. Στην συνέχεια αναφέρονται οι προκλήσεις οι οποίες έπρεπε να αντιμετωπιστούν κατά των σχεδιασμό και την ανάπτυξη των Δικτύων Τέταρτης Γενιάς.

1. Κινητός Σταθμός, όπου ομαδοποιούνται η προκλήσεις που αφορούν τα τερματικά, την ανακάλυψη ασύρματων δικτύων καθώς και την επιλογή του καταλληλότερου ασύρματου δικτύου.
2. Σύστημα, όπου υπάρχουν δύο ζητήματα που έπρεπε να επιλυθούν. Αρχικά πρέπει να γίνεται σωστή διαχείριση της θέσης όπου ανιχνεύεται μία συσκευή καθώς επίσης και να γίνεται σωστή διαχείριση των πληροφοριών που αφορούν τις τερματικές συσκευές.

3. Υπηρεσίες, όσον αφορά τις παρεχόμενες υπηρεσίες, πρέπει να επιλυθούν τα θέματα για χρήση τους από πολλαπλούς παρόχους και των υφιστάμενων μοντέλων χρέωσης.

1.2 Τεχνολογίες Δικτύων Τέταρτης Γενιάς

Τα δίκτυα Τέταρτης Γενιάς στηρίζονται στις τεχνολογίες Wi-MAX και LTE Advanced.

Η τεχνολογία WiMAX λειτουργεί με τον ίδιο τρόπο που λειτουργεί το Wi-fi όμως εξασφαλίζει πολύ μεγαλύτερη εμβέλεια επικοινωνίας. Δηλαδή ένας χρήστης θα μπορεί να χρησιμοποιήσει τη σύνδεσή στο διαδίκτυο οπουδήποτε ακόμα και αν είναι εν κινήσει αφού η εμβέλεια του θα είναι πολύ μεγαλύτερη. Συνεπώς οι εταιρίες έχουν την δυνατότητα να σχεδιάσουν το δικό τους δίκτυο με μεγάλη ευκολία καθώς δεν απαιτείται η ύπαρξη καλωδίων σε κάθε περιοχή κάτι που σημαίνει ότι θα αυξηθεί ο ανταγωνισμός προς όφελος φυσικά του πελάτη.

Η τεχνολογία LTE-Advanced αποτελεί τη πρόοδο του 3G. Χρησιμοποιεί τις τεχνολογίες MIMO (Multiple Input Multiple Output) και OFDM (Orthogonal Frequency Division Multiplexing). Η τεχνολογία MIMO συνδυάζει πολλαπλές εισόδους και πολλαπλές εξόδους πολλαπλασιάζοντας την ικανότητα μετάδοσης διαφορετικών σωμάτων από πολλαπλές κεραίες ενώ η τεχνολογία OFDM παρέχει ορθογώνια πολυπλεξία διαίρεσης συχνότητας με την οποίο χωρίζεται το κανάλι επικοινωνίας σε μεγάλο αριθμό υποκαναλιών παρέχοντας πιο αξιόπιστη επικοινωνία στις υψηλές ταχύτητες.

Πλεονεκτήματά της χρήσης του LTE-Advanced είναι οι αυξανόμενες ταχύτητες έως και 1gbps για κατέβασμα αρχείων και 500 mbps για ανέβασμα αρχείων καθώς επίσης η αποτελεσματικότητα του φάσματος, που είναι 3 φορές μεγαλύτερη από το LTE.

1.3 Πρωτόκολλα Επικοινωνίας Τέταρτης Γενιάς.

Κάθε διεπαφή ενός ασύρματου δικτύου συνδέεται μέσω πρωτοκόλλων, τα οποία χρησιμοποιούν τα στοιχεία του δικτύου και τα μηνύματα σηματοδοσίας. Τα πρωτόκολλα επικοινωνίας χωρίζονται σε δύο επίπεδα, τα πρωτόκολλα στο επίπεδο χρήστη όπου χειρίζονται δεδομένα που ενδιαφέρουν τον χρήστη και τα πρωτόκολλα στο επίπεδο ελέγχου όπου χειρίζονται μηνύματα σηματοδοσίας που ενδιαφέρουν μόνο τα στοιχεία του δικτύου.

Υπάρχουν τρία είδη πρωτοκόλλων, τα πρωτόκολλα σηματοδότησης όπου ορίζουν την γλώσσα με την οποία μπορούν να ανταλλάσσουν μηνύματα σηματοδοσίας μεταξύ τους δύο συσκευές, τα πρωτόκολλα επιπέδου χρήστη τα οποία χειρίζονται τα δεδομένα στο επίπεδο χρήστη και τα βασικά πρωτόκολλα μεταφοράς όπου μεταφέρουν τα δεδομένα και τα μηνύματα σηματοδοσίας.

1.4 Ασφάλεια Δικτύων Τέταρτης Γενιάς

Η έννοια της ασφάλειας παίζει σημαντικό ρόλο σε κάθε ανθρώπινη δραστηριότητα πόσο μάλλον στην επικοινωνία. Στην εποχή της ψηφιακής πληροφορίας με τον όρο ασφάλεια ορίζουμε κάθε προσπάθεια προστασίας των πληροφοριών από κακόβουλες ή μη εξουσιοδοτημένες πράξεις.

Μαζί με την εξέλιξη των δικτύων επικοινωνίας αυξανόταν και η ανάγκη για πιο ασφαλείς συνθήκες επικοινωνίας. Οι υπηρεσίες που παρέχονται πλέον μέσω των κινητών δικτύων, όπως οι τραπεζικές συναλλαγές, η αποστολή ευαίσθητων προσωπικών δεδομένων, οι ηλεκτρονικές αγορές τα καθιστούν ως εύκολο στόχο για πιθανές κακόβουλες πράξεις.

1.5 Παράμετροι Σχεδιασμού Ασφαλείας

Η ασφάλεια των ασύρματων δικτύων αφορά τους μηχανισμούς προστασίας τους από εξωτερικές επιθέσεις κακόβουλων χρηστών όμως βρίσκουν τον τρόπο να παραβιάζουν τα συστήματα ασφάλειας. Για το λόγο αυτό αναπτύχθηκαν ολοκληρωμένοι μηχανισμοί που παρέχουν μεγαλύτερη ασφάλεια στο σύνολο των δικτύων. Ο σχεδιασμός της ασφάλειας των δικτύων Τέταρτης Γενιάς σχεδιάστηκε με βάση τις παρακάτω παραμέτρους.

- Συνεχής διαθεσιμότητα, η ροή λειτουργίας του δικτύου να μην διακόπτεται και να είναι συνεχώς διαθέσιμη στους χρήστες.
- Συμβατότητα, η μηχανισμοί ασφαλείας να εφαρμόζονται στο σύνολο των εφαρμογών που υποστηρίζονται από ένα 4G δίκτυο.
- Χαμηλό κόστος.
- Ποιότητα υπηρεσιών, οι μηχανισμοί ασφαλείας πρέπει να ακολουθούν τις απαιτήσεις του συστήματος σε σχέση με την ποιότητα των υπηρεσιών .

1.6 Αρχιτεκτονική του Συστήματος Ασφαλείας

Ένα δίκτυο Τέταρτης Γενιάς (4G) υποστηρίζει πολλά διαφορετικά στοιχεί και παρόχους υπηρεσιών συνεπώς καθίστανται πιο ευάλωτο στις επιθέσεις. Για την προστασία των δεδομένων και των χρηστών εισάγονται πιο ευέλικτοι και βελτιωμένοι μηχανισμοί ασφαλείας.

Η Ένωση Διεθνών Τηλεπικοινωνιών (ITU) ανέπτυξε την οδηγία X.805 η οποία λειτουργεί με προσέγγιση κάθε στοιχείου χωριστά παρέχοντας συνολική ασφάλεια στο δίκτυο. Η ασφάλεια παρέχεται από κάθε από τα 8 στοιχεία χωριστά εξασφαλίζοντας την αντίσταση του δικτύου στις επιθέσεις. Οι 8 διαστάσεις ασφαλείας περιγράφονται ως εξής:

i. Έλεγχος πρόσβασης, μόνο σε εγκεκριμένους χρήστες επιτρέπεται η πρόσβαση στους πόρους του συστήματος.

ii. Πιστοποίηση, επιτρέπεται η πρόσβαση στο δίκτυο μέσω του ελέγχου της ταυτότητας των χρηστών.
iii. Μη αποκήρυξη, ελέγχονται τα δεδομένα που λαμβάνονται ελέγχοντας την εγκυρότητα της πηγής των πληροφοριών.
iv. Εμπιστευτικότητα, απαγορεύεται η διάδοση μη πιστοποιημένων δεδομένων μέσα στο δίκτυο.
v. Ασφάλεια επικοινωνίας, η επικοινωνία μέσω του δικτύου γίνεται μόνο από πιστοποιημένους χρήστες.
vi. Ακεραιότητα δεδομένων, τα δεδομένα προστατεύονται από χρήση η οποία δεν είναι πιστοποιημένη καθώς ένας μη εξουσιοδοτημένος χρήστης δεν μπορεί να αλλοιώσει τα δεδομένα.
vii. Διαθεσιμότητα, το δίκτυο όπως και οι πόροι του βρίσκονται στην διάθεση μόνο των πιστοποιημένων χρηστών.
viii. Προστασία προσωπικών δεδομένων, εξασφαλίζεται η προστασία των δεδομένων του κάθε χρήστη από κακόβουλες επιθέσεις.

1.7 Συναρτήσεις Ασφαλείας

Με βάση την αρχιτεκτονική του μηχανισμού ασφαλείας, στην συνέχεια παρουσιάζονται και αναλύονται οι διαδικασίες - λειτουργίες οι οποίες αποτελούν τα δομικά στοιχεία του μηχανισμού ασφαλείας MILENAGE. Ο MILENAGE αποτελεί τον μηχανισμό ασφαλείας ο οποίος σχεδιάζεται και υλοποιείται στην παρούσα πτυχιακή εργασία.

Αρχικά αναλύεται η διαδικασία πιστοποίησης και διευθέτησης κλειδιών καθώς και τα δεδομένα τα οποία αποτελούν την Πεντάδα Πιστοποίησης που εξασφαλίζουν την ασφάλεια του μηχανισμού όπως επίσης και η πιστοποίηση ταυτότητας του δικτύου. Στην συνέχεια αναφέρεται η λειτουργία πιστοποίησης ταυτότητας συνδρομητή μαζί με την κρυπτογράφηση των δεδομένων χρήστη και σηματοδοσίας. Επίσης, η λειτουργία πιστοποίησης γνησιότητας δεδομένων σηματοδοσίας, η διαδικασία του επανασυγχρονισμού και ο αλγόριθμος κρυπτογράφησης Rijndael αναφέρονται και αναλύονται.

Τέλος, παρουσιάζεται ο μηχανισμός ασφαλείας MILENAGE στο σύνολο του και ο τρόπος με τον οποίο γίνεται η αντιστοίχιση των λειτουργιών με τους αλγορίθμους από του οποίους αποτελείται ο μηχανισμός.

1.7.1 Δεδομένα Πεντάδας Πιστοποίησης

Οι μηχανισμοί ασφάλειας των δικτύων Τέταρτης Γενιάς κάνουν χρήση των πεντάδων πιστοποίησης AV (Authentication Vectors). Οι πεντάδες αποτελούνται από δεδομένα τα οποία εξασφαλίζουν την ασφάλεια των συστημάτων. Το πιο ευαίσθητο δεδομένο αποτελεί το Κλειδί Πιστοποίησης K του κάθε συνδρομητή.

Ποιο αναλυτικά τα δεδομένα τις κάθε πεντάδας πιστοποίησης παρουσιάζονται στην συνέχεια.

- Η Τυχαία Πρόκληση (RAND) 128 bits μήκους, η οποία χρησιμοποιείται από την λειτουργία της πιστοποίησης της ταυτότητας συνδρομητή. Πρόκειται για έναν ψευδοτυχαίο αριθμό που παράγεται από την ίδια την συσκευή χωρίς να έχει καθορισμένη τιμή.
- Η Αναμενόμενη Απάντηση (RES), μέρος της τιμής RAND μήκους 32 – 128 bits.
- Το Κλειδί Κρυπτογράφησης (CK) 128 bits μήκους, ο οποίος πρόκειται για έναν μυστικό τυχαίο αριθμό που χρησιμοποιείται για την κρυπτογράφηση των δεδομένων.
- Την Ένδειξη Πιστοποίησης (AUTN) 128 bits μήκους, η οποία περιέχει τον Κώδικα Πιστοποίησης Μηνύματος Πιστοποίησης (MAC-A) μήκους 64 bits, τον Διαδοχικό Αριθμό (SQN) μήκους 48 bits και το Πεδίο Διαχείρισης της Πιστοποίησης (AMF) 16 bits μήκους.
- Το Κλειδί Γνησιότητας (IK), μήκους 128 bits.

1.7.2 Πιστοποίηση και Διευθέτηση Κλειδιών

Την στιγμή που το Κέντρο Πιστοποίησης του Οικείου Δικτύου λάβει αίτηση πιστοποίησης ενός συνδρομητή, αποστέλλει στο Δίκτυο Εξυπηρέτησης μία πεντάδα, όπου περιέχονται όλα τα δεδομένα για την διαδικασία της Πιστοποίησης και Διευθέτησης Κλειδιών. Το Κέντρο Πιστοποίησης δημιουργεί μια νέα πεντάδα για κάθε αίτηση πιστοποίησης και στην συνέχεια την αποστέλλει στο Δίκτυο Εξυπηρέτησης έτσι ώστε να τις χρησιμοποιεί στις επόμενες διαδικασίες πιστοποίησης του κάθε συνδρομητή ξεχωριστά.

Για να συγκαλυφθεί η τιμή του SQN χρησιμοποιείται το Κλειδί Ανωνυμίας (AK), δηλαδή σε περίπτωση που ένας τρίτος καταφέρει και υποκλέψει ένα Διαδοχικό Αριθμό ενός συνδρομητή μπορεί να εντοπιστεί. Το AK αποτελεί ένα κλειδί το οποίο είναι γνωστό μόνο στην κάρτα USIM και στο Κέντρο Πιστοποίησης. Για τον υπολογισμό του χρησιμοποιείται ο αλγόριθμος **f5** με εισόδους το RAND και το K.

1.7.3 Πιστοποίηση Ταυτότητας Δικτύου

Η λειτουργία της πιστοποίησης της ταυτότητας του δικτύου δεν προβλεπόταν για τα GSM συστήματα, με αυτόν τον τρόπο δηλαδή ένας τρίτος έχει την δυνατότητα με την χρήση ενός εξομοιωτή Σταθμού Βάσης να παραπλανήσει τον Κινητό Σταθμό πως αποτελεί μέρος του Δικτύου Εξυπηρέτησης.

Με την συγκεκριμένη λειτουργία η κάρτα USIM γνωρίζει πως η τιμή RAND (τυχαία πρόκληση) την οποία λαμβάνει στέλνετε πραγματικά από το Οικείο Δίκτυο Εξυπηρέτησης καθώς επίσης αναγνωρίζει το πόσο νέο είναι το μήνυμα το οποίο λαμβάνει, δηλαδή αν έχει χρησιμοποιηθεί ξανά. Για την υλοποίηση της λειτουργίας

χρησιμοποιείτε ο αλγόριθμος **f1**, με την ύπαρξη 2 SQNs (μετρητές διαδοχικών αριθμών), ένας για το Οικείο Δίκτυο Εξυπηρέτησης και ένας για τον Κινητό σταθμό με διαφορετικό SQN για κάθε συνδρομητή. Τελικά παράγετε μία τιμή για το MAC-A, δηλαδή ένας κώδικας πιστοποίησης μηνύματος.

1.7.4 Πιστοποίηση Ταυτότητας Συνδρομητή

Η λειτουργία της πιστοποίησης της ταυτότητας του συνδρομητή, το K (κλειδί πιστοποίησης) συνεχίζει να έχει μήκος 128bits το οποίο είναι μοναδικό γα κάθε συνδρομητή. Στέλνετε από το Κέντρο Πιστοποίησης του Οικείου Δικτύου μία RAND, μήκους 128bits στον Κινητό Σταθμό. Στην συνέχεια η κάρτα υπολογίζει την απάντηση χρησιμοποιώντας την τιμή RAND και K με την χρήση του αλγορίθμου **f2**, παράγοντας την τιμή RES, δηλαδή την Απάντηση Χρήστη**.**

1.7.5 Κρυπτογράφηση Δεδομένων Χρήστη και Σηματοδοσίας

Με την εξέλιξη των δικτύων τέταρτης γενιάς παρέχεται μεγαλύτερη ασφάλεια στα δεδομένα τα οποία παράγονται από τον χρήστη όπως επίσης και στα δεδομένα σηματοδοσίας τα οποία μεταδίδονται στον δίαυλο επικοινωνίας. Αυτό επιτυγχάνεται με την αύξηση του μήκους των κλειδιών που χρησιμοποιούνται στα 128 bits.

Στη λειτουργιά κρυπτογράφησης των δεδομένων η κάρτα USIM χρησιμοποιεί το K (κλειδί πιστοποίησης) μαζί με την RAND, την οποία λαμβάνει από το Δίκτυο Εξυπηρέτησης, και παράγει το κλειδί κρυπτογραφίας CK (Cipher Key). Για την υλοποίηση χρησιμοποιείτε ο αλγόριθμος **f3.**

1.7.6 Πιστοποίηση Γνησιότητας Δεδομένων Σηματοδοσίας

Για να μην υπάρχει κίνδυνος κάποιος κακόβουλος χρήστης να έχει την δυνατότητα να αποστείλει μήνυμα σηματοδοσίας στην κάρτα USIM και κατά συνέπεια να εξαπατηθεί ο συνδρομητής, στα δίκτυα Τέταρτης Γενιάς ο συνδρομητής παράλληλα με την πιστοποίηση της ταυτότητά του, πιστοποιεί και την ταυτότητα του δικτύου. Έχει προστεθεί επίσης και η λειτουργία της πιστοποίησης της γνησιότητας των δεδομένων σηματοδοσίας, τα οποία προέρχονται είτε από τον Κινητό Σταθμό είτε από το Δίκτυο Εξυπηρέτησης.

Για την υλοποίηση της λειτουργίας είναι βασικό το IK (Integrity Key) μήκους 128 bits, το οποίο υπολογίζεται από το RAND και το K του Οικείου Δικτύου. Ο αλγόριθμος που χρησιμοποιείται είναι ο **f4.** Το IK υπολογίζεται ταυτόχρονα από την κάρτα USIM και από το Οικείο Δίκτυο και αποστέλλετε στο Δίκτυο Εξυπηρέτησης.

1.7.7 Διαδικασία Επανασυγχρονισμού

Στην λειτουργία Πιστοποίησης Ταυτότητας Δικτύου υπάρχει περίπτωση η τιμή SQN να μην συμπεριλαμβάνεται στα ανεκτά όριο του Κινητού Σταθμού. Σε αυτή την περίπτωση αποστέλλεται στο Κέντρο Πιστοποίησης ένα μήνυμα αποτυχίας συγχρονισμού η οποία περιλαμβάνει την μεταβλητή AUTS. Στην AUTS υπάρχει ο Διαδοχικός Αριθμός του Κινητού Σταθμού (SQN_{MS}) και ο Κώδικας Πιστοποίησης Μηνύματος Συγχρονισμού (MAC-S). Ο Διαδοχικός Αριθμός προστατεύεται με την χρήση του AK (Κλειδί Ανωνυμίας), το οποίο υπολογίζετε με την χρήση του αλγόριθμου **f5*** με εισόδους τη RAND και το K (Κλειδί Πιστοποίησης).

Ο Κώδικας Πιστοποίησης Μηνύματος Συγχρονισμού υπολογίζεται με την χρήση του αλγορίθμου **f1*.** Οι είσοδοι του αλγορίθμου είναι ο SQN_{MS} RAND, το AMF και το K.

Οι συναρτήσεις του επανασυγχρονισμού δεν φανερώνουν κανένα στοιχείο για τις υπόλοιπες λειτουργίες πουν χρησιμοποιούντα στην διαδικασία Πιστοποίησης και Διευθέτησης Κλειδιών.

1.7.8 Κρυπτογράφηση Δεδομένων

Ο αλγόριθμος Rijndael, ο οποίος θα αναλυθεί στο επόμενο κεφάλαιο, έχει ορισθεί ως βασικός πυρήνας (Εκ) του μηχανισμού ασφαλείας και χρησιμοποιείται γα την κωδικοποίηση των δεδομένων, παρέχοντας την ασφάλειά τους. Ως είσοδοι του αλγορίθμου ορίζονται ένα καθαρό κείμενο (plaintext) και το κλειδί κρυπτογράφησης (cipher key), μήκους 128 bits το καθένα, και έξοδος το κωδικοποιημένο κείμενο (cipher text). Ο αλγόριθμος Rijndael μπορεί να αντικατασταθεί από οποιονδήποτε αλγόριθμο κωδικοποίησης ανάλογα με τις απαιτήσεις του σχεδιασμού του κάθε μηχανισμού ασφαλείας.

1.7.9 Διαδικασία Πιστοποίησης και Διευθέτησης Κλειδιών - MILENAGE

Επτά διαφορετικοί αλγόριθμοι χρησιμοποιούνται για τον υπολογισμό των δεδομένων που απαιτούνται κατά την λειτουργία του μηχανισμού ασφαλείας. Όλα τα δίκτυα δεν είναι απαραίτητο να χρησιμοποιούν τις ίδιες λειτουργίες. Για αυτό τον λόγο έχει ορισθεί ένα σύνολο αλγορίθμων, το οποίο και ακολουθούμε κατά την διεκπεραίωση και πτυχιακής εργασίας, γνωστό ως MILENAGE.

Ο κάθε αλγόριθμος που περιλαμβάνει ο MILENAGE υλοποιεί κάθε μία από τις λειτουργίες που αναφέρθηκαν στις προηγούμενες παραγράφους οι οποίες παρουσιάζονται ονομαστικά μαζί με τις αντίστοιχες εξόδους στην συνέχεια.

- f1: Αλγόριθμος Πιστοποίησης Ταυτότητας Δικτύου. (MAC-A - Network Authenticatio Code)
- f1*: Αλγόριθμος Πιστοποίησης Μηνύματος Συγχρονισμού. (MAC-S - Resynchronisation Authentication Code)

- f2: Αλγόριθμος Υπολογισμού Απάντησης Χρήστη. (RES - Response to Challenge)
- f3: Αλγόριθμος Υπολογισμού Κλειδιού Κρυπτογράφησης. (CK - Cipher Key)
- f4: Αλγόριθμος Υπολογισμού Κλειδιού Γνησιότητας. (IK - Integrity Key)
- f5: Αλγόριθμος Υπολογισμού Κλειδιού Ανωνυμίας. (AK - Anonymity Key)
- f5*: Αλγόριθμος Υπολογισμού Κλειδιού Ανωνυμίας στην Διαδικασία Επανασυγχρονισμού. (AK)

Σημείωση: Επειδή η χρήση του Κλειδιού Ανωνυμίας δεν είναι υποχρεωτική, σε περίπτωση που ο διαχειριστής του δικτύου δεν επιθυμεί να την χρησιμοποιήσει τότε οι αλγόριθμοι f5 και f5 θέτονται ίσοι με το μηδέν.*

1.8 Κάρτα USIM

Η κάρτα USIM (UMTS Subscriber Identity Module) αποτελεί ένα μέρος της ασφάλειας των δικτύων Τέταρτης Γενιάς και πρόκειται για την εξέλιξη της κάρτας SIM της οποίας γινόταν χρήση στα δίκτυα Τρίτης Γενιάς. Η κάρτα USIM παρέχει αναβαθμισμένες διαδικασίες και υπηρεσίες σε σχέση με την κάρτα SIM.

Σε κάθε κάρτα είναι αποθηκευμένο το Κλειδί Πιστοποίησης, η αντίστοιχη Παγκόσμια Ταυτότητα Συνδρομητή (IMSI) καθώς και οι αλγόριθμοι οι οποίοι χρησιμοποιούνται στην λειτουργία του μηχανισμού ασφαλείας. Για αυτόν τον λόγο πρέπει να είναι προστατευμένη συνεχώς από οποιαδήποτε επίθεση και να παρέχει ασφάλεια στα δεδομένα τα οποία περιέχει.

Για μεγαλύτερη ασφάλεια γίνεται η χρήση του PIN (Personal Identity Number), ο οποίος είναι ο Προσωπικός Αριθμός Αναγνώρισης, και είναι απαραίτητος για να έχει κάποιος πρόσβαση στην κάρτα. Αν γίνει τρεις φορές εισαγωγή του αριθμού PIN, τότε η κάρτα κλειδώνεται και δεν είναι δυνατή η πρόσβαση του χρήστη στο δίκτυο.

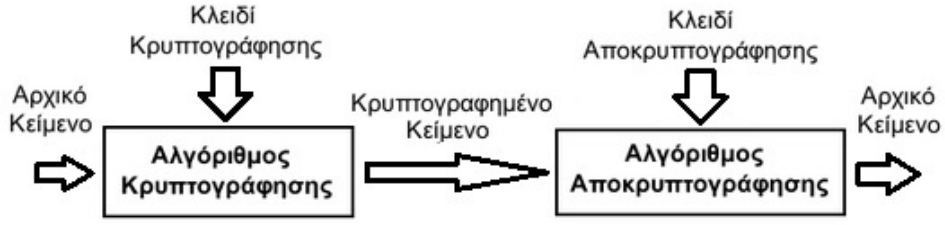
ΚΕΦΑΛΑΙΟ 2
ΑΛΓΟΡΙΘΜΟΙ ΚΡΥΠΤΟΓΡΑΦΗΣΗΣ

2.1 Κρυπτογραφία

Ως κρυπτογραφία έχει ορισθεί ο κλάδος που ασχολείται με την μετατροπή της πληροφορίας με σκοπό την εξασφάλιση του απορρήτου. Διασφαλίζεται η ιδιωτικότητα των πληροφοριών με το να κρατιούνται τα δεδομένα κρυφά από μη εγγεκριμένους χρήστες. Η κρυπτογραφία αποτελεί ένα εργαλείο για την ασφάλεια των πληροφοριών, δηλαδή προστατεύονται οι πληροφορίες ως προς την ακεραιότητα, εμπιστευτικότητα και διαθεσιμότητα. Εκτελείται μέσω του αλγόριθμου κρυπτογράφησης ο οποίος μετατρέπει τα δεδομένα σε μη αναγνωρίσιμα. Κατά την αποκρυπτογράφηση τα κρυπτογραφημένα δεδομένα μετατρέπονται στην αρχική τους μορφή. Για λειτουργία της κρυπτογράφησης είναι απαραίτητη η χρήση ενός μυστικού κλειδιού.

Συνεπώς, η κρυπτογράφηση των δεδομένων του χρήστη είναι απαραίτητη τόσο για την αποτροπή υποκλοπών των συνομιλιών όσο και για την προστασία ευαίσθητων δεδομένων, όπως για παράδειγμα τηλεφωνικοί αριθμοί, κωδικοί πρόσβασης και ευαίσθητα προσωπικά στοιχεία.

Ιστορικά η κρυπτογραφία χρησιμοποιήθηκε για την κρυπτογράφηση των μηνυμάτων με την μετατροπή της πληροφορίας που περιείχαν σε έναν γρίφο που χωρίς την γνώση του κρυφού τρόπου μετασχηματισμού θα παρέμενε μη κατανοητός και ασφαλές. Οι πρώτες μορφές κρυπτογράφησης υλοποιούνταν πάνω στην γλωσσική δομή ενώ στις νεότερες μορφές γίνεται χρήση που μαθηματικού υπόβαθρου εισάγοντας τις έννοιες των διακριτών μαθηματικών, την θεωρία των αριθμών, την στατιστική, την υπολογιστική πολυπλοκότητα και την συνδυαστική ανάλυση.



Εικόνα 2.1 Διαδικασία Κρυπτογράφησης και Αποκρυπτογράφησης

2.1.1 Απαιτήσεις Κρυπτογραφίας

Για την ανάπτυξη ενός αλγορίθμου κρυπτογράφησης ορίζονται οι παρακάτω απαιτήσεις:
Εμπιστευτικότητα: Η προστασία των δεδομένων ενάντια σε μη εξουσιοδοτημένους χρήστες η οποία υλοποιείται μέσω μηχανισμών ελέγχου πρόσβασης και μέσω της κωδικοποίησης τους.
Ακεραιότητα: Η προστασία των δεδομένων ενάντια σε κακόβουλες τροποποιήσεις και αντικαταστάσεις.
Πιστοποίηση: Η επιβεβαίωση της ταυτότητας ενός χρήστη και της πηγής αποστολής.
Μη Άρνηση Αποδοχής: Ο αποστολέας και ο παραλήπτης της πληροφορίας δεν μπορούν να αρνηθούν την αυθεντικότητα της μετάδοσης και την δημιουργίας της πληροφορίας.

2.1.2 Βασικές Έννοιες Κρυπτογραφίας

Στην συνέχεια αναφέρονται και αναλύονται βασικές έννοιες που αφορούν την κρυπτογραφία.

- Κρυπτογράφηση, η διαδικασία μετασχηματισμού ενός μηνύματος σε μία μη κατανοητή μορφή με την χρήση ενός κρυπτογραφικού αλγορίθμου.
- Αποκρυπτογράφηση, η αντίστροφη διαδικασία της κρυπτογράφησης, δηλαδή από το κρυπτογραφημένο κείμενο παράγεται η αρχική πληροφορία.
- Κρυπτογραφικός αλγόριθμος (cipher), η μέθοδος μετασχηματισμού των δεδομένων σε μία μορφή η οποία δεν επιτρέπει την αποκάλυψη του περιεχομένου σε μη εξουσιοδοτημένους χρήστες
- Αρχικό κείμενο (plaintext), το μήνυμα το οποίο αποτελεί την είσοδο σε ένα κρυπτογραφικό αλγόριθμο.
- Κλειδί (key), ένα σύνολο από bit που χρησιμοποιείται ως είσοδος σε έναν κρυπτογραφικό αλγόριθμο.
- Κρυπτογραφημένο Κείμενο (ciphertext), η έξοδος ενός αλγορίθμου κρυπτογράφησης.

2.2 Είδη Αλγορίθμων Κρυπτογραφίας

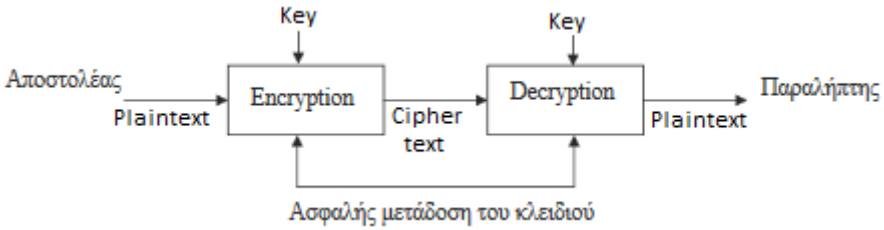
2.2.1 Συμμετρικοί Αλγόριθμοι

Συμμετρικοί είναι οι αλγόριθμοι οι οποίοι χρησιμοποιούν το ίδιο κλειδί για κρυπτογράφηση και αποκρυπτογράφηση. Από τους πιο γνωστούς είναι ο αλγόριθμος DES (Data Encryption Standard).

Στους συμμετρικούς αλγόριθμους ο αποστολέας και ο παραλήπτης του μηνύματος χρησιμοποιούν ένα κοινό κλειδί. Ο αποστολέας κρυπτογραφεί το μήνυμα με βάση αυτό το κλειδί και ο παραλήπτης το αποκρυπτογραφεί με βάση το ίδιο κλειδί. Για να

γνωρίζουν και ο αποστολέας και ο παραλήπτης το κλειδί θα πρέπει αρχικά να έχει γίνει ανταλλαγή.

Με αυτόν τον τρόπο όμως υπάρχει μεγάλος κίνδυνος να υποκλαπεί το κλειδί από κάποιον τρίτο που παρακολουθεί τις γραμμές επικοινωνίας ή και να διαρρεύσει από το ένα από τα δύο μέρη. Χρησιμοποιώντας έναν συμμετρικό αλγόριθμο θα πρέπει το κλειδί να παραμένει κρυφό, κάτι που είναι εξαιρετικά δύσκολο στα ανοικτά δίκτυα με πολλούς χρήστες, όπως για το Internet.



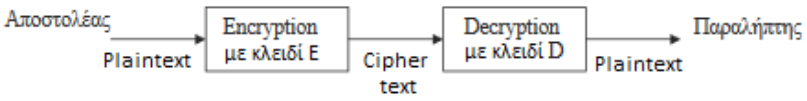
Εικόνα 2.2 Διαδικασία συμμετρικής κρυπτογράφησης

2.2.2 Ασύμμετροι Αλγόριθμοι

Ασύμμετροι είναι οι αλγόριθμοι οι οποίοι χρησιμοποιούν ένα ζευγάρι κλειδιών για κρυπτογράφηση, το δημόσιο κλειδί, το οποίο κρυπτογραφεί τα στοιχεία, και ένα αντίστοιχο ιδιωτικό κλειδί (μυστικό) το οποίο χρησιμοποιείται κατά την διαδικασία της αποκρυπτογράφησης.

Το δημόσιο κλειδί γίνεται γνωστό ενώ το ιδιωτικό κλειδί μένει κρυφό. Ο καθένας δηλαδή μπορεί να κρυπτογραφήσει το κείμενο, όμως μόνο ο κάτοχος του ιδιωτικού κλειδιού μπορεί να το αποκωδικοποιήσει.

Η τεχνολογία της ασύμμετρης κρυπτογραφίας, βάσει συγκεκριμένων αλγορίθμων, παράγει τυχαία ζεύγη κρυπτογραφικών κλειδιών τα οποία χαρακτηρίζονται από δύο σημαντικές ιδιότητες. Αρχικά, το κάθε κλειδί κρυπτογραφεί ψηφιακά δεδομένα τα οποία μπορούν να αποκρυπτογραφηθούν μόνο από το άλλο κλειδί, και επίσης δεν είναι δυνατό, με τις παρούσες δυνατότητες της τεχνολογίας, να συμπεράνει κανείς το ένα κλειδί όταν γνωρίζει το άλλο.



Εικόνα 2.3 Διαδικασία ασύμμετρης κρυπτογράφησης

2.3 Μηχανισμοί Κρυπτογράφησης

Οι κρυπτογραφικοί αλγόριθμοι μπορούν να χωριστούν σε δύο διαφορετικές κατηγορίες με βάση τον τρόπο κρυπτογράφησης των μηνυμάτων, στην κρυπτογράφηση τμήματος (block) και κρυπτογράφηση ροής (stream).

2.3.1 Κωδικοποίηση Τμήματος (Block)

Ο μηχανισμός κρυπτογράφησης Block είναι ένας μηχανισμός συμμετρικής κρυπτογράφησης που μετατρέπει ένα μη κρυπτογραφημένο μπλοκ το οποίο δεν έχει καθορισμένο μήκος κειμένου (plaintext), σε ένα κρυπτογραφημένο μπλοκ με ίδιο μήκος κειμένου (ciphertext). Η κρυπτογράφηση υλοποιείται με την χρήση ενός μυστικού κλειδιού που χορηγείται από τον χρήστη. Η αποκρυπτογράφηση υλοποιείται με την εφαρμογή του αντίστροφου μηχανισμού στο κρυπτογραφημένο κείμενο χρησιμοποιώντας το ίδιο μυστικό κλειδί. Το καθορισμένο μήκος καλείται block size και για πολλούς κρυπτογραφικούς αλγορίθμους έχουν ορισθεί τα 64 bits.

Η κρυπτογράφηση Block λειτουργούν επαναληπτικά, κάνοντας κρυπτογράφηση ένος μπλοκ διαδοχικά αρκετές φορές. Σε κάθε γύρο, ο ίδιος μετασχηματισμός εφαρμόζεται στα δεδομένα χρησιμοποιώντας ένα υπο-κλειδί. Με ειδική συνάρτηση υπολογίζεται το σύνολο των υπο-κλειδιών, τα οποία προέρχονται από το μυστικό κλειδί. Ο αριθμός των επαναλήψεων της επαναληπτικής κρυπτογράφησης εξαρτάται από το επίπεδο της ασφάλειας και την απόδοση του συστήματος.

2.3.2 Κωδικοποίηση Ροής (Stream)

Ο μηχανισμός κρυπτογράφησης Stream είναι ένας μηχανισμός συμμετρικής κρυπτογράφησης, ο οποίος είναι πολύ πιο γρήγορος από έναν Block . Σε αντίθεση με του αλγόριθμους κρυπτογράφησης τμήματος, οι οποίοι υλοποιούνται με μεγάλα κομμάτια δεδομένων, τα μπλοκ, οι αλγόριθμοι ροής υλοποιούνται με μικρότερες μονάδες απλού κειμένου (plaintext). Η κρυπτογράφηση ενός κειμένου με έναν αλγόριθμο μπλοκ όταν χρησιμοποιείται το ίδιο κλειδί θα έχει πάντα το ίδιο αποτέλεσμα ενώ με την χρήση ενός αλγόριθμου ροής, ο μετασχηματισμός των μικρότερων κομματιών θα διαφέρει κατά την διάρκεια της κρυπτογράφησης.

Παράγεται μια ακολουθία από bits που χρησιμοποιείται ως κλειδί, το keystream. Η κρυπτογράφηση επιτυγχάνεται με τον συνδυασμό του keystream με το plaintext, συνήθως μέσω της πράξης XOR.

2.4 Αλγόριθμοι κρυπτογράφησης

Οι αλγόριθμοι κρυπτογράφησης αποτελούνται από μία σειρά βημάτων επεξεργασίας των δεδομένων που μετατρέπουν το αρχικό κείμενο (plaintext) σε ένα κρυπτογραφημένο κείμενο (cipher text). Οι παράμετροι που χρησιμοποιούνται από έναν αλγόριθμο κρυπτογράφησης προκύπτουν από ένα μυστικό κλειδί (cipher key). Γνωστοί αλγόριθμοι κρυπτογράφησης είναι οι παρακάτω:

DES (Data Encryption Standard)

Πρόκειται για ένα συμμετρικό αλγόριθμο. Αναπτύχθηκε από την IBM, την NSA και το National Institute of Standards and Technology (NIST). Χρησιμοποιεί κλειδί 64 bits από τα οποία τα 8 αποτελούν bits ισοτιμίας. Ο DES εκτός για την κρυπτογράφηση χρησιμοποιείται για στην παραγωγή MACs, δηλαδή των κωδικών πιστοποίησης.

Triple-DES(Data Encryption Standard)

Πρόκειται για μία παραλλαγή του DES. Το μήνυμα κρυπτογραφείται και αποκρυπτογραφείται διαδοχικά με διαφορετικά κλειδιά. Τα επιπλέον κλειδιά δημιουργούνται από το κοινό μυστικό κλειδί με κατάλληλο αλγόριθμο. Με αυτό τον τρόπο ενισχύουμε τον βασικό αλγόριθμο. Υπάρχουν τέσσερις μέθοδοι υλοποίησης.

- DES-EEE3 (Encrypt – Encrypt - Encrypt): Υλοποιούνται τρεις επαναλαμβανόμενες κρυπτογραφήσεις με τρία διαφορετικά κλειδιά.
- DES-EDE3 (Encrypt – Decrypt - Encrypt): Το κείμενο κρυπτογραφείται στην συνέχεια αποκρυπτογραφείται και τέλος κρυπτογραφείται. Για τις παραπάνω διαδικασίες χρησιμοποιούνται τρία διαφορετικά κλειδιά.
- DES-EEE2: Ίδια με την πρώτη διαδικασία με την διαφορά ότι χρησιμοποιούνται δύο διαφορετικά κλειδιά.
- DES-EDE2: Ίδια με την δεύτερη διαδικασία με την διαφορά ότι χρησιμοποιούνται δύο διαφορετικά κλειδιά.

DESX

Πρόκειται για μία ακόμα παραλλαγή του DES. Η είσοδος και η έξοδος της κρυπτογράφησης στο DESX περνάει από μια πράξη XOR με ένα κλειδί 64 bits.

IDEA (International Data Encryption Algorithm)

Πρόκειται για έναν αλγόριθμο κρυπτογράφησης τμημάτων μεγέθους 64 bits και με 128 bits κλειδιά. Η διαδικασία της κρυπτογράφησης απαιτεί 8 σύνθετες επαναλήψεις. Εφαρμόζεται και σε υλικό (hardware) και σε λογισμικό (software). Αποτελεί μία λιγότερο αποδοτική λύση σε σχέση με άλλους αλγόριθμους.

RC2

Πρόκειται για έναν αλγόριθμο κρυπτογράφησης τμημάτων με κλειδί μεταβλητού μήκους. Έχει μέγεθος ίσο με 64 bits και είναι τέσσερις φορές γρηγορότερος από τον DES

RC4

Πρόκειται για αλγόριθμο κρυπτογράφησης ροών, stream cipher. Έχει μεταβλητό μήκος κλειδιού και λειτουργεί σε επίπεδο byte.

RC5

Πρόκειται για αλγόριθμο κρυπτογράφησης τμημάτων με κλειδί μεταβλητού μήκους. Έχει μέγεθος του κάθε μπλοκ ίσο με 32, 64, 128 bits. Ο αριθμός των επαναλήψεων μπορεί να είναι από 0 έως και 255.

SHA και SHA-1 (Secure Hash Algorithm)

Πρόκειται για αλγόριθμους που αναπτύχθηκαν από το NIST και αποτελεί επανέκδοση του SHA. Με τον SHA-1 διορθώθηκε μια ατέλεια του SHA και λαμβάνει σαν είσοδο ένα κείμενο μικρότερο από 264 bits μήκους και παράγει μία σύνοψη 162 bits. Αποτελεί έναν αλγόριθμο κατακερματισμού και δημοσίου κλειδιού.

AES (Advanced Encryption Standard)

Πρόκειται για έναν συμμετρικό αλγόριθμο κρυπτογράφησης τμημάτων, ο οποίος μέχρι τώρα αποτελεί έναν από τους πιο δημοφιλείς αλγόριθμους κρυπτογραφίας. Στην συνέχεια γίνεται περαιτέρω ανάλυση για τον AES.

Καθώς ο αλγόριθμος DES δεν παρέχει πλέον την ασφάλεια που είναι αναγκαία για δίκτυα νέας τεχνολογίας, αποφασίστηκε πως έπρεπε να αντικατασταθεί. Ο TDES αν και είναι αρκετά ασφαλής και θα είναι και για τα επόμενα χρόνια, μειονεκτεί στο ότι είναι αργός σε λειτουργίες με χρήση λογισμικού καθώς εκτελεί τρεις φορές περισσότερους γύρους από τον DES.

Για αυτούς τους λόγους το National Institute of Standards and Technology (NIST), προκήρυξε διαγωνισμό για την χρήση νέου κρυπτογραφικού προτύπου εμπορικής χρήσης, του AES.

Στον διαγωνισμό επικράτησε ο αλγόριθμος Rijndael καθώς πληρούσε τα κριτήρια συγκριτικής αξιολόγησης τα οποία εντάχθηκαν στις τρεις κατηγορίες οι οποίες παρουσιάζονται στην συνέχεια.

- Ασφάλεια αλγορίθμων: αξιολογείται η αντοχή των αλγορίθμων σε επιθέσεις και η σύγκριση της ασφάλειας του κάθε αλγορίθμου σε σχέση με τους υπόλοιπους.

Οι υποψήφιοι αλγόριθμοι θα έπρεπε να περιλαμβάνουν πιο αποδοτικά και ευέλικτα χαρακτηριστικά ασφάλειας.

- Κόστος: αξιολογούνται οι απαιτήσεις της μνήμης και της υπολογιστικής ισχύος του κάθε αλγορίθμου, καθώς και το κόστος της προστασίας των πνευματικών δικαιωμάτων ιδιοκτησίας.
- Απλότητα: αξιολογείται η απλότητα, η ευελιξία και η παροχή συμπληρωματικών κρυπτογραφικών λειτουργιών παρέχοντας μεγαλύτερη ασφάλεια.

Δέκα πέντε αλγόριθμοι είχαν γίνει αποδεκτοί στην αρχή και στην συνέχεια επιλέχθηκαν πέντε από αυτούς, οι αλγόριθμοι MARS, Serpent, RC6, Rijndael και Twofish. Αυτός που τελικά επιλέχθηκε επισήμως ως AES ο αλγόριθμος Rijndael, ο οποίος είχε υποβληθεί από τους Βέλγους κρυπτογράφους J. Daemen και V. Rijmen.

2.5 Αλγόριθμος Κρυπτογράφησης Rijndael

Ο Rijndael αποτελεί έναν αλγόριθμο ο οποίος εκτελεί μία συμμετρική λειτουργία κρυπτογράφησης 128, 192 και 256-bits κλειδιών. Το ότι πρόκειται για έναν συμμετρικό αλγόριθμο σημαίνει ότι το κλειδί κρυπτογράφησης μπορεί να υπολογιστεί από το αντίστοιχο αποκρυπτογράφησης και αντιστρόφως ή μπορεί τα κλειδιά να είναι ίδια. Η ασφάλεια ενός συμμετρικού αλγόριθμου βασίζεται στο κλειδί, το οποίο πρέπει να παραμένει μυστικό.

Η λειτουργία είναι επαναληπτική, δηλαδή σε κάθε μπλοκ γίνεται μια επεξεργασία η οποία επαναλαμβάνεται ανάλογα με το μήκος κλειδιού, όπου κάθε επανάληψη ονομάζεται γύρος (round). Στο αρχικό γύρο έχουμε σαν είσοδο ένα κείμενο (plaintext), ενώ στους υπόλοιπους γύρους σαν είσοδο έχουμε την ακολουθία από bits που έχει παραχθεί από τον προηγούμενο γύρο. Ακόμα είσοδος είναι και το κλειδί (K) το οποίο έχει παραχθεί από το κλειδί που έχει οριστεί σαν είσοδος βασισμένο σε κάποια διαδικασία που ορίζει ο αλγόριθμος (KeySchedule). Σαν έξοδο λαμβάνουμε το κρυπτογραφημένο κείμενο (ciphertext) ίδιου μεγέθους με το plaintext.

Η κύρια μονάδα επεξεργασίας είναι το byte. Δηλαδή τα bits των εισόδων ή των κλειδιών χωρίζονται σε ομάδες των 8-bytes. Όλες η λειτουργίες που εκτελεί ο αλγόριθμος γίνονται σε ένα πίνακα δύο διαστάσεων ο οποίος ορίζεται ως Κατάσταση (State). Αποτελείται από τέσσερις γραμμές από bytes και κάθε γραμμή να αποτελείται από 4 bytes.

2.6 Ανάλυση Αλγορίθμου Rijndael

Ο αλγόριθμος Rijndael αποτελεί έναν επαναληπτικό αλγόριθμο τον οποίο διακρίνει η απλότητα καθώς συνδυάζει αντικαταστάσεις και αναδιατάξεις των κελιών επαναλαμβανόμενα. Το μπλοκ εισόδου είναι 128-bit ενώ το κλειδί κρυπτογράφησης μπορεί να έχει μήκος 128, 192 ή 256-bits (AES-128, AES-192 ή AES-256 αντίστοιχα) με τον AES-128 έχει να έχει επικρατήσει. Ανάλογα με το μήκος του κλειδιού κρυπτογράφησης εκτελούνται και οι αντίστοιχοι γύροι κρυπτογράφησης οι οποίο δίνονται στην συνέχεια.

Πόσοι γύροι (rounds) ανά μήκος κλειδιού
AES-128: 9 γύροι κρυπτογράφησης + 1 τελικός γύρος
AES-192: 11 γύροι κρυπτογράφησης + 1 τελικός γύρος
AES-256: 13 γύροι κρυπτογράφησης + 1 τελικός γύρος

Κατά της διαδικασία της κρυπτογράφησης, κάθε γύρος αποτελείται από μετασχηματισμούς σε επίπεδο byte. Οι κυκλικές συναρτήσεις οι οποίες εκτελούνται σε κάθε γύρο κρυπτογράφησης είναι οι παρακάτω.

- SubByte, μπλοκ αντικατάστασης
- ShiftRow, μπλοκ ολίσθησης,
- MixColumns, μπλοκ ανάμειξης,
- KeyAdd, μπλοκ πρόσθεσης κλειδιού.

Το κλειδί κρυπτογράφησης (cipher key) που είναι η είσοδος του αλγορίθμου είναι το κλειδί που προστίθεται στο κείμενο εισόδου (plaintext) πριν ξεκινήσει η διαδικασία της κρυπτογράφησης. Σε κάθε γύρο υπάρχει ένα στάδιο στο οποίο προστίθεται στο μπλοκ ένα κλειδί, το οποίο όμως δεν είναι το cipher key αλλά κάποιο κλειδί το οποίο έχει παραχθεί από την διαδικασία Επέκτασης Κλειδιούς (KeySchedule). Δηλαδή, σε κάθε γύρο χρησιμοποιείται ένα διαφορετικό κλειδί (round key) που έχει παραχθεί από την λειτουργία KeySchedule με είσοδο το κλειδί κρυπτογράφησης.

2.6.1 Ψευδοκώδικας αλγορίθμου

Κατά τον σχεδιασμό του αλγορίθμου κρυπτογράφησης ορίζουμε τον με τον παρακάτω ψευδοκώδικα ο οποίος παρουσιάζει αναλυτικά τις λειτουργίες καθώς και τους κύκλους επανάληψης:

```
Cipher (byte_in [4*Nb], byte_out [4*Nb], byte_key[4*Nb]
          word w [Nb*Nr+1] )

begin
          byte State [4*Nb]
          state = in
          KeySchedule(key,w);
          KeyAdd (state, w [o, Nb-1])
          For round=1 step 1 to Nr-1
             ByteSub (state)
             ShiftRows (state)
             MixColumns (state)
             KeyAdd (state, w[round*Nb, (round+1)*Nb-1])
         end for
         ByteSub (state)
         ShiftRows (state)
         KeyAdd (state, w [Nr*Nb, (Nr+1)*Nb-1])
         out = state
```

end

Σημείωση 1: ο πίνακας w χρησιμοποιείται για να δηλώσει τα κλειδιά του κάθε γύρου (round keys) τα οποία παράγονται από την διαδικασία KeySchedule.
Σημείωση 2: το Nr εξαρτάται το bits αλγορίθμου που έχουμε επιλέξει για την υλοποίηση (128, 192, 256 bits).

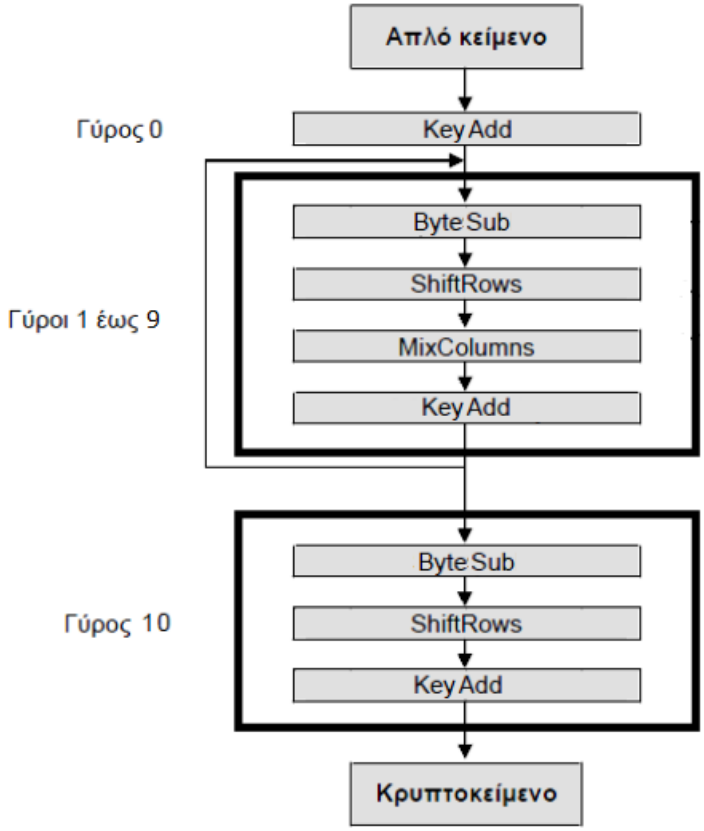
Συνοψίζοντας, ο αλγόριθμος λαμβάνει σαν είσοδο το plaintext. Με την χρήση των παραγόμενων κλειδιών τα οποία παράγονται από την διαδικασία KeySchedule και την χρήση των λειτουργιών ByteSub, ShiftRows, MixColumns και KeyAdd εξάγεται το κρυπτογραφημένο κείμενο (cipher text).

Η εσωτερική δομή του αλγορίθμου αναπαριστάται με την εξής σειρά υλοποιήσεων της κάθε λειτουργίας:

- Αρχική υλοποίηση της λειτουργίας KeyAdd (γύρος 0)

- 9 γύροι, από το 1 ως το 9, οι οποίοι αποτελούνται από:
    - Υλοποίηση της λειτουργίας ByteSub
    - Υλοποίηση της λειτουργίας ShiftRows
    - Υλοποίηση της λειτουργίας MixColumns
    - Υλοποίηση της λειτουργίας KeyAdd

- Τελευταίος γύρος (γύρος 10) ο οποίος αποτελείται από:
    - Υλοποίηση της λειτουργίας ByteSub
    - Υλοποίηση της λειτουργίας ShiftRows
    - Υλοποίηση της λειτουργίας KeyAdd



Εικόνα 2.4 Λειτουργίες του αλγορίθμου Rijndael

2.7 Λειτουργίες του Αλγορίθμου Κρυπτογράφησης

Ο αλγόριθμος αποτελείται από μία σειρά από λειτουργίες και καθώς εκτελείται υλοποιούνται οι γύροι που μετατρέπουν την είσοδο σε έξοδο. Τα ενδιάμεσα αποτελέσματα ονομάζονται Κατάσταση (State) και αναπαριστούνται ως δισδιάστατοι πίνακες byte [4x4], με σύνολο 128 bits.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Εικόνα 2.5 Απεικόνιση του πίνακα Κατάστασης (State)

Στην συνέχεια αναφέρονται και αναλύονται όλες οι λειτουργιές του αλγορίθμου Rijndael οι οποίες εκτελούνται σε κάθε γύρο κρυπτογράφησης.

2.7.1 Λειτουργία KeyAdd

Η λειτουργία KeyAdd αποτελεί έναν αλγόριθμο κρυπτογράφησης ροής. Κάθε byte από το μήνυμα εισόδου συνδυάζεται, υλοποιείται δηλαδή η πράξη XOR με το κλειδί της επανάληψης, το οποίο παράγεται χρησιμοποιώντας την λειτουργία KeyShedule. Η λειτουργία KeyAdd είναι η μόνη που χρησιμοποιεί το κλειδί και αυτό γίνεται για την μεγαλύτερη ασφάλεια των δεδομένων. Η πράξη XOR γίνεται κατά στήλες, δηλαδή μεταξύ των 4 bytes μιας στήλης του State και του round key.

2.7.2 Λειτουργία ByteSub

Η λειτουργία ByteSub αποτελεί μία μη-γραμμική πράξη, όπου κάθε byte του μηνύματος εισόδου, αντικαθίσταται με ένα άλλο byte σύμφωνα με το Πίνακα Αντικατάστασης (S-box) όπου η αντικατάσταση γίνεται ανεξάρτητα σε κάθε byte.

$$State\_out(i) = S\text{-}box[State\_in(i)]$$

2.7.2.1 Πίνακας Αντικατάστασης S-box

Ο πίνακας Αντικατάστασης, S-box, ο οποίος είναι ένας αντιστρέψιμος πίνακας 16x16 από bytes, αποτελεί το μόνο τμήμα του αλγορίθμου που είναι μη γραμμικό και συνεπώς καθορίζει σε πολύ μεγάλο βαθμό την ασφάλεια του. Είναι ειδικά σχεδιασμένος ώστε να προκαλεί σύγχυση καθώς η σχέση του bit εισόδου και του bit εξόδου είναι ιδιαίτερα πολύπλοκη.

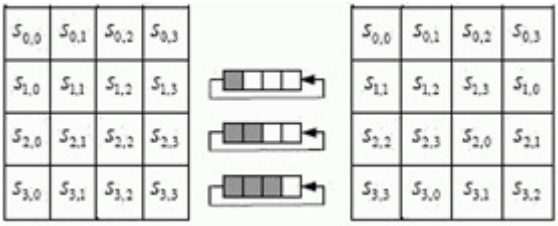
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Εικόνα 2.6 Ο πίνακας αντικατάστασης S-BOX

2.7.3 Λειτουργία ShiftRows

Στην λειτουργία ShiftRows γίνεται ολίσθηση των bytes ανά τετράδες του μηνύματος εισόδου κυκλικά προς τα αριστερά για έναν συγκεκριμένο αριθμό επαναλήψεων. Στην πρώτη γραμμή δεν γίνεται ολίσθηση, η δεύτερη γραμμή ολισθάνει κατά ένα byte, η τρίτη γραμμή κατά δύο bytes και η τέταρτη γραμμή κατά τρία bytes.

- Γραμμή 0: καμία μετατόπιση
- Γραμμή 1: μετατόπιση κατά 1 byte
- Γραμμή 2: μετατόπιση κατά 2 bytes
- Γραμμή 3: μετατόπιση κατά 3 bytes

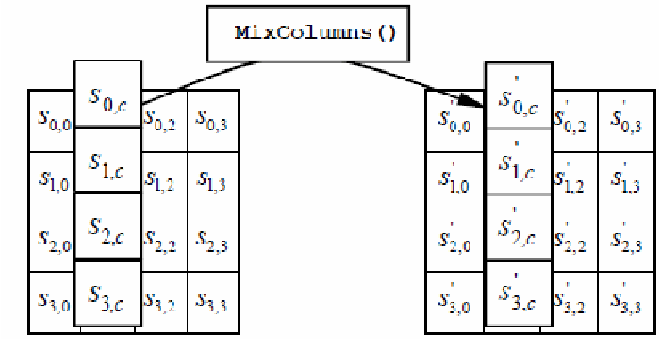


Εικόνα 2.7 Η λειτουργία της ολίσθησης

2.7.4 Λειτουργία MixColumns

Η λειτουργία MixColumns αποτελεί έναν γραμμικό μετασχηματισμό ο οποίος ενεργεί σε κάθε σειρά του πίνακα State. Αν κάθε στήλη του πίνακα αναπαρασταθεί ως πολυώνυμο 3ου βαθμού, τότε η διαδικασία MixColumns είναι ένας πολλαπλασιασμός με το πολυώνυμο c(x) ο οποίος ακολουθείται από μια αναγωγή με το x4+1.

$$s(x)_i = s_{3,i}x^3 + s_{2,i}x^2 + s_{1,i}x + s_{0,i}$$



Εικόνα 2.8 Η λειτουργία του μετασχηματισμού

2.7.5 Λειτουργία Επέκτασης Κλειδιού - KeySchedule

Για την ολοκλήρωση των 10 γύρων για υλοποίηση του αλγορίθμου Rijndael χρειάζονται 10 κλειδιά (round keys) τα οποία παράγονται από την λειτουργία KeySchedule. Σε κάθε γύρο ένα από τα παραγόμενα κλειδιά χρησιμοποιείται ως είσοδος στην λειτουργία KeyAdd.

Η λειτουργία επέκτασης κλειδιού (KeySchedule) λαμβάνει ως είσοδο το 128-bit κλειδί κρυπτογράφησης, cipher key και κάθε φορά που απαιτείται από τον αλγόριθμο ένα round key, εξάγεται ένα επεκταμένο κλειδί. Το αρχικό round key, το οποίο

χρησιμοποιείται στον αρχικό γύρο, είναι το ίδιο το cipher key και κάθε καινούργιο κλειδί παράγεται από το προηγούμενο.

2.8 Αλγόριθμος Αποκρυπτογράφησης

Οι λειτουργίες του αλγορίθμου κρυπτογράφησης μπορούν να αντιστραφούν και με αυτή την διαδικασία οι λειτουργίες τοποθετούνται σε αντίστροφη σειρά έτσι ώστε να υλοποιηθεί ένας μηχανισμός ο οποίος θα αποκρυπτογραφεί το κρυπτογραφημένο κείμενο που έχει παραχθεί από τον αλγόριθμο Rijndael. Όπως και στην διαδικασία της κρυπτογράφησης, στην αποκρυπτογράφηση υπάρχουν τέσσερις λειτουργίες που χρησιμοποιούνται, η InvShiftRows, η InvSubBytes, η InvMixColumns και η AddRoundKey. Επίσης αντί του πίνακα S-box χρησιμοποιείται ο αντίστροφός του, ο Inverse S-box.

2.9 Ασφάλεια Αλγορίθμου Rijndael

Ο Rijndael αποτελεί έναν ασφαλή, αποδοτικό και ευέλικτο αλγόριθμο για την κρυπτογράφηση πληροφοριών και για αυτό επιλέχθηκε ανάμεσα στους 15 υποψήφιους αλγόριθμους ως AES.

Η ασφάλειά του υπολογίζεται από το γεγονός ότι η υλοποίηση με 128-bit κλειδί θα υπήρχαν 2^{128} κλειδιά, συνεπώς αν χρησιμοποιούσαμε 1 δις παράλληλους επεξεργαστές που ο καθένας θα εκτελούσε τον έλεγχο ενός κλειδιού ανά 1 psec θα χρειαζόμασταν 10 δις εκατομμύρια χρόνια για να ελέγξουμε όλα τα κλειδιά. Με βάση την σημερινή τεχνολογία, ο αλγόριθμος Rijndael εκτιμάται ότι επαρκεί για τα επόμενα 20 χρόνια.

Η σχεδίαση του αλγορίθμου Rijndael έγινε εξαρχής ώστε να εξασφαλιστεί η αντοχή του στο σύνολο των επιθέσεων. Πραγματοποιήθηκαν δηλαδή δοκιμές ασφαλείας και για τα παρακάτω είδη επιθέσεων.

- Επίθεση κόλουρων διαφορικών
- Επιθέσεις κορεσμού
- Επίθεση Gilbert-Minier
- Επιθέσεις παρεμβολής
- Αναζήτηση αδυνάμων κλειδιών
- Επιθέσεις σχετιζομένων κλειδιών.

ΚΕΦΑΛΑΙΟ 3
ΥΛΟΠΟΙΗΣΗ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

3.1 Μέθοδοι Υλοποίησης Ολοκληρωμένων Κυκλωμάτων

Ολοκληρωμένο Κύκλωμα (IC - Integrated Circuit) ονομάζεται ένα κύκλωμα συνδεδεμένων λογικών πυλών τα οποία δημιουργούνται πάνω σε φύλλα ημιαγωγών, κατά κύριο λόγο πυριτίου, τα οποία ονομάζονται τσιπ. Ολοκληρωμένα κυκλώματα χρησιμοποιούνται σε κάθε στοιχείο ηλεκτρονικού συστήματος.

Για τον σχεδιασμό ενός ολοκληρωμένου κυκλώματος χρειάζονται αποδοτικές σχεδιαστικές μέθοδοι ώστε να καλυφθούν οι ανάγκες για ιεραρχικό σχεδιασμό, αυτοματισμό και την επαναχρησιμοποίηση σχεδιασμών. Για αυτό τον λόγο υπάρχουν δύο μέθοδοι υλοποίησης Ολοκληρωμένων Κυκλωμάτων, το ASIC και το FPGA.

Στην παρούσα πτυχιακή εργασία θα κάνουμε χρήση ενός ολοκληρωμένου κυκλώματος για να υλοποιήσουμε τον μηχανισμό ασφαλείας MILENAGE.

3.2 Υλοποίηση με χρήση ASICs

Το ASIC (Application Specific Integration Circuit), δηλαδή ολοκληρωμένο κύκλωμα συγκριμένης εφαρμογής, αποτελεί ένα κύκλωμα το οποίο υλοποιείται για να λειτουργεί σε συγκεκριμένες εφαρμογές χωρίς την δυνατότητα περαιτέρω μορφοποίησης του σχεδιασμού ή της λειτουργίας του. Το γεγονός ότι λαμβάνουν την τελική τους μορφή κατά την κατασκευή τους και ότι τα χαρακτηριστικά τους πρέπει να είναι καθορισμένα από την αρχή τα καθιστά μη ευέλικτα.

Οι εταιρίες κατασκευής κυκλωμάτων για να αποφύγουν την πώληση ελαττωματικών προϊόντων στους χρήστες τα οποία στην συνέχεια δεν θα μπορούν να διορθώσουν, δοκιμάζουν τα κυκλώματα πρώτα σε FPGA με σκοπό να ελέγξουν την ορθή λειτουργία τους και στην συνέχεια για την τελική κατασκευή και παραγωγή προγραμματίζουν το ολοκληρωμένο κύκλωμα ASIC.

Η υλοποίηση με την χρήση ASIC παρέχει κυκλώματα βέλτιστα χωροταξικά και ενεργειακά κυκλώματα καθώς ένα κύκλωμα ASIC περιέχει μόνο τα στοιχεία, της πύλες και τα κανάλια τα οποία απαιτεί ο σχεδιασμός του κυκλώματος και με αυτόν τον τρόπο δεν χρησιμοποιείται περισσότερη ενέργεια και υλικό για να παραχθούν τα αποτελέσματα.

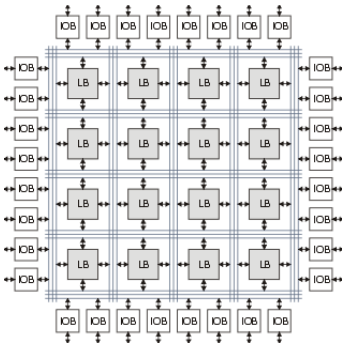
3.3 Υλοποίηση με χρήση FPGAs

Το FPGA (Field Programmable Gate Array), δηλαδή επιτόπια συστοιχία προγραμματιζόμενων πυλών, είναι ένα προγραμματιζόμενο ολοκληρωμένο κύκλωμα γενικής χρήσης το οποίο διαθέτει έναν πολύ μεγάλο αριθμό από τυποποιημένες πύλες, απαριθμητές, καταχωρητές μνήμης και άλλες ψηφιακές λειτουργίες. Σε ορισμένα από αυτά ενσωματώνονται και αναλογικές λειτουργίες. Ανάλογα με το μηχανισμό των οποίο θέλουμε να υλοποιήσουμε μπορούμε να προγραμματίσουμε

κατάλληλα το FPGA έτσι ώστε να χρησιμοποιούνται συγκεκριμένες μονάδες και να διασυνδέονται μεταξύ του με σκοπό της λειτουργίας του ως ολοκληρωμένο κύκλωμα. Η χρήση του είναι κατάλληλη για μηχανισμούς που οι παράμετροι λειτουργίας αλλάζουν συνεχώς ή για παραγωγή κυκλωμάτων σε μικρές ποσότητες.

Βασική δομική μονάδα του FPGA αποτελεί το μπλοκ, το οποίο περιέχει χιλιάδες λογικές πύλες οι οποίες συνδέονται μεταξύ τους μέσω διακλαδωτών και καλωδιώσεων. Κάποια από αυτά τα μπλοκ λειτουργούν ως σήματα εισόδου και εξόδου. Κατά τον προγραμματισμό ενός FPGA σε κάθε λογικό μπλοκ ορίζεται μια συγκεκριμένη λειτουργία για να εκτελέσει. Η λειτουργία είναι ανάλογη με τον σχεδιασμό που έχει κάνει ο σχεδιαστής και επίσης ενεργοποιούνται οι κατάλληλες διασυνδέσεις για την ένωση τους.

Πιο συγκεκριμένα τα FPGAs τα οποία κατασκευάζει η εταιρεία Xilinx αποτελούνται από slices και αποτελούν μικρότερες δομές των λογικών μπλοκ που ονομάζονται CLB (Configurable Logic Block).



Εικόνα 3.1 Δομή FPGA

Ένα FPGA μπορεί να αγοραστεί ή ως μεμονωμένο chip όπου ο χρήστης πρέπει να το τοποθετήσει σε πλακέτα ή ως πιο ολοκληρωμένη λύση, δηλαδή τα τσιπ να είναι ήδη τοποθετημένα σε πλακέτα. Οι πλακέτες διαθέτουν το σύστημα προγραμματισμού του FPGA καθώς και μία πληθώρα διεπαφών που μπορεί να προγραμματίσει ο χρήστης. Η χρήση των FPGAs υπερέχει σε σχέση με συμβατικές υλοποιήσεις καθώς ο χρήστης μπορεί να το προγραμματίσει ανεξάρτητα της εταιρίας παραγωγής του FPGA και των εξαρτημάτων, επίσης οι εταιρίες ανάπτυξης των FPGA έχουν δημιουργήσει κομμάτια υλικού τα οποία χρησιμοποιούν οι χρήστες και παρέχουν διευκόλυνση και αύξηση της ποιότητας του σχεδιασμού. Τέλος αποτελούν μία ιδανική λύση ανάμεσα στις υλοποιήσεις λογισμικού και της χρήση των ASIC.

Το κυριότερο πλεονέκτημα ενός FPGA είναι το χαμηλό κόστος. Επίσης μπορούμε εύκολα να επανασχεδιάσουμε ένα κύκλωμα και να το υλοποιήσουμε με άλλες ρυθμίσεις μέχρις ότου να ανταποκρίνεται στις αρχικές προδιαγραφές. Μειονεκτεί όμως σε σχέση με την ταχύτητα, στο πλήθος των πυλών τα οποία χρησιμοποιούνται αλλά και στην κατανάλωση ισχύος.

Για τον προγραμματισμό ενός FPGA απαιτείται η συγγραφή του κώδικα σε γλώσσα περιγραφής υλικού. Ο κώδικας με την χρήση ειδικού λογισμικού μεταφράζεται σε δεδομένα διαμόρφωσης FPGA, τα firmwares. Επίσης, ο προγραμματισμός υλοποιείται σε επίπεδο λογικών πυλών και ψηφιακών εξαρτημάτων και παρέχεται η δυνατότητα στον σχεδιαστή να χειρίζεται άμεσα το υλικό του κυκλώματος.

3.3.1 Ιστορική Αναδρομή FPGA

Πρόγονος της τεχνολογία FPGA αποτελούν οι προγραμματιζόμενες μνήμες (PROM) καθώς και οι προγραμματιζόμενες λογικές διατάξεις (PLDs). Παρόλο που οι παραπάνω τεχνολογίες παρείχαν την δυνατότητα να προγραμματιστούν στο πεδίο λειτουργίας, οι λογικές πύλες ήταν συνδεδεμένες με την σχεδιαστική λογική μόνιμα.

Το πρώτο κύκλωμα με επαναπρογραμματιζόμενες πύλες σχεδιάστηκε την δεκαετία του 1980. Στην δεκαετία του 1990 υπήρξε αύξηση των πωλήσεων και συνεπώς αύξηση της πολυπλοκότητας των κυκλωμάτων και η τεχνολογία των FPGA πλέον έχει κατακτήσει πολλούς τομείς όπως οι τηλεπικοινωνίες και τα δίκτυα αλλά και οι αυτοκινητοβιομηχανία και διάφορα προϊόντα.

3.3.2 Χαρακτηριστικά του FPGA

Τα ιδιαίτερα χαρακτηριστικά ενός FPGA παρουσιάζονται και αναλύονται στην συνέχεια.

- Με την διακοπή της τάσης τροφοδοσίας του FPGA χάνεται ο προγραμματισμός του, συνεπώς απαιτείται ένας εξωτερικός επεξεργαστής ή μία μνήμη με μόνιμη συγκράτηση δεδομένων από τα οποία προγραμματίζεται εκ νέου κάθε φορά που επανέρχεται η τάση τροφοδοσίας.
- Ο προγραμματισμός ενός FPGA μπορεί εύκολα να τροποποιηθεί κάθε φορά που ο σχεδιαστής αλλάξει το λογισμικό του επεξεργαστή ή τα δεδομένα μνήμης.
- Ένα FPGA μπορεί να επαναπρογραμματιστεί απεριόριστες φορές.
- Συγκριτικά με τα ASIC, στο FPGA παρέχεται μεγαλύτερη κατανάλωση ισχύος.

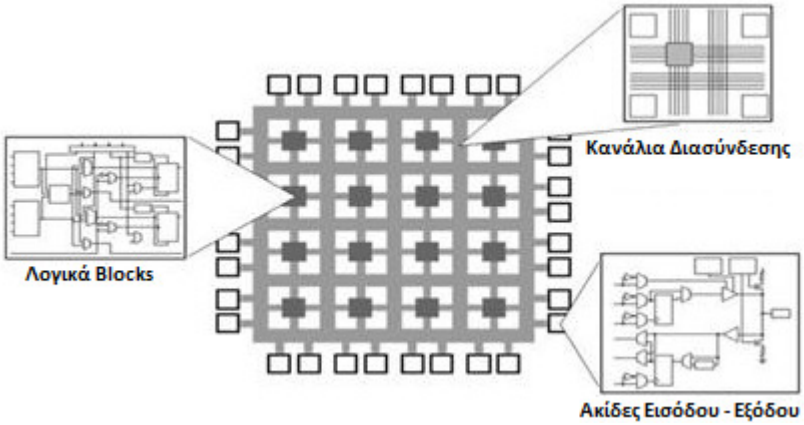
3.3.3 Λειτουργία του FPGA

Το FPGA μπορεί να εκτελέσει πολλές λειτουργίες μετά τον προγραμματισμό του, ανάλογα με τον τρόπο σχεδιασμού του. Αποτελείται από βασικά λογικά κυκλώματα όπως πύλες, πολυπλέκτες, αντιστροφείς, αθροιστές και flip-flops αλλά και πιο πολύπλοκων μονάδων όπως μονάδες πολλαπλασιαστών, μνήμης, επεξεργαστών και ελεγκτών.

Με τον κατάλληλο προγραμματισμό ενός FPGA υλοποιούνται ο ιδανικές διασυνδέσεις των βασικών κυκλωμάτων δημιουργώντας κάθε φορά διαφορετικά κυκλώματα έχοντας με αυτόν τον τρόπο μία πιο διευρυμένη περιοχή χρήσης τους παρέχοντας μεγαλύτερη ευελιξία στους χρήστες. Οι διασυνδέσεις μεταξύ των λογικών κυκλωμάτων έχουν την μορφή διακοπτών on/off, δηλαδή αναπαριστούνται ως ανοιχτοί/κλειστοί. Κατά τον προγραμματισμό, η θέση του διακόπτη αποθηκεύεται στην μνήμη RAM.

3.3.4 Αρχιτεκτονική του FPGA

Ένα FPGA αποτελείται από έναν δισδιάστατο πίνακα από προγραμματιζόμενα λογικά μπλοκ (CLBs), τα κανάλια διασύνδεσης και τις ακίδες εισόδου-εξόδου. Ένα λογικό μπλοκ αποτελείται από πολλαπλά λογικά κελιά, που με την σειρά τους αποτελούνται από τις γεννήτριες συναρτήσεων look-up tables 6 εισόδων (LUT), την δυνατότητα κατανεμημένης μνήμης και καταχωρητή ολίσθησης, την ειδική λογική κρατουμένου για την υλοποίηση μαθηματικών πράξεων και του πολυπλέκτες για πιο αποδοτικά κυκλώματα.



Εικόνα 3.2 Αρχιτεκτονική FPGA

3.3.4.1 Λογικό Μπλοκ

Τα λογικά μπλοκ είναι η κύριοι λογικοί πόροι για την υλοποίηση τόσο της ακολουθιακής όσο και συνδυαστικής λογικής. Κάθε λογικό μπλοκ περιέχει δύο slices τα οποία συνδέονται σε ένα πίνακα με διακόπτες παρέχοντάς τους την πρόσβαση στο δίκτυο διασύνδεσης. Το βασικό μπλοκ για την υλοποίηση των κυκλωμάτων αποτελούν οι γεννήτριες συναρτήσεων (LUT). Για τις εισόδους μιας λογικής πράξης δημιουργείται ένας πίνακας αλήθειας. Για κάθε είσοδο ο πίνακας αλήθειας περιγράφει την τιμή της εξόδου της συνάρτησης.

Τα slices χρησιμοποιούνται για τις λογικές και τις αριθμητικές πράξεις του κυκλώματος και για τις λειτουργίες της ROM και το καθένα αποτελείται από τέσσερις γεννήτριες λογικών συναρτήσεων (LUTs), οχτώ στοιχεία μνήμης και

πολυπλέκτες. Επίσης μερικά slices, τα οποία ονομάζονται SLICEM, παρέχουν την δυνατότητα υλοποίησης δύο ακόμα λειτουργιών όπως η ολίσθηση δεδομένων και η αποθήκευση των δεδομένων μέσω RAM. Τα slices που δεν παρέχουν τις δύο επιπλέον λειτουργίες καλούνται SLICEL. Το κάθε μπλοκ μπορεί να περιέχει ένα SLICEM και ένα με δύο SLICEL.

Η υλοποίηση των γεννητριών συναρτήσεων (LTP) υλοποιούνται με LUT 6 εισόδων όπου περιέχουν έξι ανεξάρτητες εισόδους και δύο ανεξάρτητες εξόδους για κάθε μία από τις τέσσερις γεννήτριες λογικών συναρτήσεων του κάθε slice. Επίσης σε κάθε slice υπάρχουν 8 στοιχεία αποθήκευσης. Τέσσερα από αυτά μπορούν να διαμορφωθούν ή ως D flip-flops ή ως μανδαλωτές.

3.3.4.2 Κανάλια Διασύνδεσης

Η διασύνδεση μεταξύ των λογικών μπλοκ με σκοπό την διακλάδωση διαδρομών δεδομένων και την δρομολόγηση των σημάτων σε όλο το κύκλωμα παρέχεται μέσω ενός πλέγματος διατάξεων διακοπτών. Το σύστημα διασύνδεσης παρέχει του εξής τύπους συνδέσεων γενικού σκοπού.

- Γραμμές μονού μήκους,
- Γραμμές διπλού μήκους,
- Μακρές γραμμές.

Με την χρήση γραμμών μονού μήκους παρέχεται ένα πλέγμα από οριζόντιες και κατακόρυφες γραμμές οι οποίες διασύνδεουν μια διάταξη από το σύνολο των διακοπτών. Οι διακόπτες παρέχουν έναν μικρό αριθμό συνδέσεων μεταξύ των διαδρομών του σήματος μέσα σε κάθε σύνολο αλλά δεν παρέχουν σύνδεση με κανένα καθολικό διακόπτη. Οι απευθείας γραμμές διασύνδεσης παρέχουν δρομολόγηση μεταξύ των γειτονικών κατακόρυφων και οριζοντίων μπλοκ που ανήκουν στην ίδια γραμμή ή στήλη, οι οποίες αποτελούν τοπικές συνδέσεις με υψηλές ταχύτητες.

Οι γραμμές διπλού μήκους διασχίζουν την απόσταση δύο μπλοκ, παρακάμπτοντας τα τυχόν ενδιάμεσα μπλοκ και στην συνέχεια εισέρχονται σε ένα πίνακα διακοπτών. Οι συγκεκριμένες γραμμές προσφέρουν μια πιο αποτελεσματική μέθοδο διασύνδεσης μη λαμβάνοντας υπόψη τις διατάξεις διακοπτών της διαδρομής και αυτό έχει ως αποτέλεσμα την καθυστέρηση της διαδρομής.

Τέλος, οι μακρές γραμμές διασχίζουν το σύνολο των μπλοκ κατακόρυφα και οριζόντια και χρησιμοποιούνται για την οδήγηση σημάτων ελέγχου με τα οποία παρέχεται η μικρή χρονική απόκλιση.

3.3.4.3 Ακίδες Εισόδου - Εξόδου

Οι ακίδες εισόδου - εξόδου (Input/Output Blocks – IOBs), που βρίσκονται περιμετρικά του ολοκληρωμένου κυκλώματος, συνδέουν τον πινάκα των λογικών μπλοκ και τις πηγές δρομολογήσεων με τους εξωτερικούς ακροδέκτες της συσκευής.

Ως στοιχεία αποθήκευσης δεδομένων στα κυκλώματα ορίζονται τα Flip-Flops. Μια έξοδος του LUT συνδέεται με την είσοδο ενός Flip-Flop. Ακόμη ένα Flip-Flop μπορεί να χρησιμοποιηθεί και ως μανδαλωτής που λειτουργεί είτε σε θετική είτε σε αρνητική λογική.

3.3.4 Προγραμματισμός του FPGA

Για τον προγραμματισμό ενός FPGA χρησιμοποιούνται ειδικές γλώσσες περιγραφής υλικού καθώς το πλεονέκτημά τους είναι ότι αποτελούν τις καταλληλότερες γλώσσες περιγραφής υλικού για τον σχεδιασμό μεγάλων και με μεγάλες απαιτήσεις υλοποιήσεις. Οι πιο διαδεδομένες γλώσσες περιγραφής υλικού είναι η VHDL και η Verilog καθώς έχει απλουστευτεί πολύ η διαδικασία του σχεδιασμού.

Τα λογικά μπλοκ προγραμματίζονται και εκτελούν τις λογικές πράξεις AND και XOR όπως επίσης και συνδυασμούς των πράξεων αυτών. Κάποια από τα λογικά μπλοκ υποκαθιστούν στοιχεία μνήμης όπως flip-flops. Δεν μπορούμε να υλοποιήσουμε πολύπλοκα κυκλώματα αλλά μπορούμε εύκολα να τα επαναπρογραμματίσουμε.

ΚΕΦΑΛΑΙΟ 4
ΣΧΕΔΙΑΣΜΟΣ ΜΗΧΑΝΙΣΜΟΥ ΑΣΦΑΛΕΙΑΣ MILENAGE

4.1 Ο Μηχανισμός MILENAGE

Ο MILENAGE είναι ένας μηχανισμός ασφάλειας και πιστοποίησης σε 3G και 4G δίκτυα, ο οποίος παρέχει αμοιβαίο έλεγχο ταυτότητας καθώς η κάρτα USIM παρέχει την ασφάλεια στην πιστοποίηση δικτύου. Ο αλγόριθμος υλοποιείται στο AUC (Κέντρο Πιστοποίησης) του χειριστή και στην κάρτα USIM. Αποτελείται από 7 λειτουργίες - αλγόριθμους πιστοποίησης.

4.2 Λειτουργίες του MILENAGE

Οι λειτουργίες πιστοποίησης και παραγωγής κλειδιών του αλγόριθμου MILENAGE περιγράφονται παρακάτω μαζί με τις αντίστοιχες εξόδους:

f1: λειτουργία πιστοποίησης ταυτότητας δικτύου,
έξοδος MAC-A - Network Authenticatio Code.
f1*: λειτουργία ελέγχου επανασυγχρονισμένου μηνύματος,
έξοδος MAC-S -Resynchronisation Authentication Code.
f2: λειτουργία πιστοποίησης χρήστη,
έξοδος RES - Response to Challenge.
f3: λειτουργία παραγωγής κλειδιού κρυπτογράφησης,
έξοδος CK - Cipher Key.
f4: λειτουργία παραγωγής κλειδιού ακεραιότητας,
έξοδος IK - Integrity Key.
f5: λειτουργία παραγωγής κλειδιού ανωνυμίας,
έξοδος AK - Anonymity Key.
f5*: λειτουργία παραγωγής κλειδιού ανωνυμίας για επανασυγχρονισμένο μήνυμα
έξοδος AK.

4.2.1 Είσοδοι – Έξοδοι αλγορίθμων

Στην συνέχεια παρουσιάζουμε αναλυτικά για κάθε αλγόριθμο τις εισόδους και τις εξόδους.

<u>Είσοδοι του αλγόριθμου f1</u>

K: Subscriber Key - Κλειδί Συνδρομητή (128-bits).
RAND: Random Challenge - Τυχαία τιμή (128-bits).
SQN: Sequence Number - Αριθμός ακολουθίας (48-bits).
AMF: Authentication management field - Πεδίο διαχείρισης ταυτοποίησης (16-bits).

Είσοδοι του αλγόριθμου f1*

K: Subscriber Key - Κλειδί Συνδρομητή (128-bits).
RAND: Random Challenge - Τυχαία τιμή (128-bits).
SQNMS: Sequence Number - Αριθμός ακολουθίας (48-bits).
AMF: Authentication management field – Πεδίο διαχείρισης ταυτοποίησης (16-bits).

Είσοδοι των αλγορίθμων f2, f3, f4, f5 και f5*

K: Subscriber Key - Κλειδί Συνδρομητή (128-bits).
RAND: Random Challenge - Τυχαία τιμή (128-bits).

Έξοδος του αλγόριθμου f1

MAC-A: Network authentication code - Κωδικός πιστοποίησης δικτύου (64-bits).

Έξοδος του αλγόριθμου f1*

MAC-S: Resynsh authentication code - Κωδικός πιστοποίησης δικτύου (64-bits).

Έξοδος του αλγόριθμου f2

RES: Response – Απόκριση (64-bits).

Έξοδος του αλγόριθμου f3

CK: Confidentiality Key - Κλειδί εμπιστευτικότητας (128-bits).

Έξοδος του αλγόριθμου f4

IK: Integrity Key – Κλειδί ακεραιότητας (128-bits).

Έξοδος του αλγόριθμου f5

AK: Anonymity Key - Κλειδί ανωνυμίας (48-bits).

Έξοδος του αλγόριθμου f5*

AK: Resynch Anonymity Key - Κλειδί ανωνυμίας για επανασυγχρονισμό (48-bits).

4.3 Τα συστατικά του MILENAGE

Ο αλγόριθμος MILENAGE αποτελείται από δύο βασικά συστατικά:

1. Ένας αλγόριθμος κρυπτογράφησης που λαμβάνει μία είσοδο κειμένου (plaintext) 128-bit και ένα κλειδί κρυπτογράφησης (cipher key) 128-bit και σαν έξοδο επιστρέφει ένα κρυπτογραφημένο κείμενο (cipher text) 128-bit.

   y= E[x]k, όπου x είναι το plaintext, k το cipher key και y το cipher text

2. Μία τιμή για την OP. Η τιμή της OP 128-bit παρέχει την δυνατότητα λειτουργίας των αλγορίθμων από διαφορετικούς χρήστες. Κάθε χρήστης επιλέγει τη τιμή του OP. Με την χρήση της OP και το K υπολογίζουμε το OPc.

4.3.1 Ορισμός της OP

Το Πεδίο Διαμόρφωσης του Αλγόριθμου κάθε διαχειριστή, OP (Operator Code), έχει μήκος 128 bits και χρησιμοποιείται για να διαχωρίζει την λειτουργικότητα των αλγορίθμων κατά την εκτέλεσή του από διαφορετικούς διαχειριστές και επίσης είναι αναγκαία για την υλοποίηση βασικών 3G και 4G αλγορίθμων, στην περίπτωσή μας ο MILENAGE. Η τιμή του Πεδίου Διαμόρφωσης του Αλγορίθμου εξαρτάται από τον διαχειριστή του δικτύου.

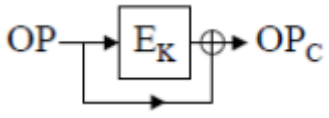
Το σύνολο των αλγορίθμων MILENAGE έχει σχεδιαστεί έτσι ώστε να είναι ασφαλές ακόμα και αν είναι δημοσίως γνωστή η τιμή του OP. Ωστόσο είναι πολύ πιο ασφαλές να κρατηθεί η τιμή του Πεδίου Διαμόρφωσης του Αλγορίθμου μυστική.

Υπολογισμός OPc

Η τιμή του OPc υπολογίζεται από το Κλειδί Πιστοποίησης (K) και το Πεδίο Διαμόρφωσης του Αλγόριθμου (OP) με τον παρακάτω τρόπο.

$$OPc = OP \oplus Ek(OP)$$

όπου Ek,
αλγόριθμος βάσης Rijndael με είσοδο την OP.



Εικόνα 4.1 Υπολογισμός OPc

Η τιμή του OPc θα πρέπει να υπολογίζεται εκτός της κάρτας USIM και ύστερα να αποθηκεύεται σε αυτή. Έτσι παρέχεται μεγαλύτερη ασφάλεια καθώς το OP δεν χρειάζεται να αποθηκεύεται στην USIM.

Αφού ο αλγόριθμος λάβει την 128bit OP υπολογίζεται η τιμή της OPc με εισόδους την τιμή της OP και το κλειδί κρυπτογράφησης K η OP δεν χρησιμοποιείται ξανά κατά την εκτέλεση των αλγορίθμων του MILENAGE.

Με αυτόν τον τρόπο έχουμε εναλλακτικές επιλογές για του εφαρμογή του MILENAGE σε ένα USIM:

1. Η OPc υπολογίζεται εκτός του USIM, δηλαδή υπολογίζεται ως μέρος του USIM και αποθηκεύεται στην κάρτα, ενώ η τιμή της OP δεν αποθηκεύεται.
2. Η OPc υπολογίζεται εντός του USIM, δηλαδή αποθηκεύεται η τιμή της OP και η OPc υπολογίζεται κάθε φορά που καλείται ένας αλγόριθμος

4.3.2 Αλγόριθμος Κρυπτογράφησης – Rijndael

Κατά τον σχεδιασμό του αλγορίθμου MILENAGE, στην παρούσα πτυχιακή εργασία, επιλέγουμε ως βασικό αλγόριθμο κρυπτογράφησης τον αλγόριθμο Rijndael (επιλεγμένος ως AES) καθώς πρόκειται για έναν ισχυρό αλγόριθμο κρυπτογράφησης. Ο Rijndael διασφαλίζει την ορθή λειτουργία και την προστασία από επιθέσεις σε μεγαλύτερο βαθμό σε σχέση με άλλους αλγορίθμους της κατηγορίας του.

Ο αλγόριθμος κρυπτογράφησης βασίζεται σε έναν πίνακα 128-bit που ονομάζεται Κατάσταση (State). Ο πίνακας State αποτελεί έναν δυσδιάστατο πίνακα με τέσσερις στήλες και 4 γραμμές (4x4), όπου κάθε κελί του πίνακα περιέχει 1 byte των 8 bit. Υλοποιείται με 9 επαναλήψεις, χωρίς την τελευταία επανάληψη υπολογίζοντας τις λειτουργίες που παρουσιάζονται στην συνέχεια. Ο αλγόριθμος Rijndael αναπαριστάται με το Ek

4.3.3 Διαδικασία Επέκτασης Κλειδιού - KeySchedule

Η λειτουργία επέκτασης κλειδιού (KeySchedule) λαμβάνει ως είσοδο το 128-bit cipher key και κάθε φορά που απαιτείται από τον αλγόριθμο ένα round key, εξάγεται ένα επεκταμένο κλειδί. Το κάθε επεκταμένο κλειδί είναι ένα 11 128-bits word.

Για την δημιουργία του πρώτου round key, το κλειδί αντιγράφεται στις 4 πρώτες λέξεις - words του επεκταμένου κλειδιού. Το υπόλοιπο round γεμίζεται ανά τέσσερις λέξεις την κάθε φορά. Για την λειτουργία της επέκτασης κλειδιού εκτελούνται οι διεργασίες RotWord, δηλαδή μία κυκλική αριστερόστροφη ολίσθηση κατά ένα byte σε μία λέξη και SubWord, δηλαδή μία αντικατάσταση ενός byte, σε κάθε byte του μετατοπισμένου αποτελέσματος, χρησιμοποιώντας το S-box.

4.4 Πλαίσιο MILENAGE

Ο μηχανισμός - αλγόριθμος MILENAGE είναι σχεδιασμένος έτσι ώστε τα συστατικά του να μπορούν να αντικατασταθούν από το κάθε χειριστή ο οποίος έχει σκοπό να σχεδιάσει έναν δικό του μηχανισμό ασφαλείας.

Για τον σχεδιασμό του αλγορίθμου, κατά την διάρκεια της διεκπεραίωσης της πτυχιακής εργασίας, ορίσαμε ένα πλαίσιο για τον αλγόριθμο MILENAGE, το οποίο δίνεται στην συνέχεια.

ΠΛΑΙΣΙΟ ΑΛΓΟΡΙΘΜΟΥ MILENAGE

Μία τιμή του **OPc** 128-bit παράγεται από την OP και το K με την εξής διαδικασία:

$$OPc = OP \oplus E[OP]k$$

Μία ενδιάμεση τιμή **TEMP** 128-bit υπολογίζεται με τον ακόλουθο τρόπο:

$$TEMP = E[RAND \oplus OPc]k$$

Μία τιμή **IN1** 128-bit κατασκευάζεται ως εξής:

$$IN1[0] .. IN1[47] = SQN[0] .. SQN[47]$$
$$IN1[48] .. IN1[63] = AMF[0] .. AMF[15]$$
$$IN1[64] .. IN1[111] = SQN[0] .. SQN[47]$$
$$IN1[112] .. IN1[127] = AMF[0] .. AMF[15]$$

Δηλαδή η τιμή IN1 κατασκευάζεται από τις τιμές SQN και AMF οι οποίες χρησιμοποιούνται σαν είσοδος στους αλγορίθμους f1 και f1*.

Πέντε σταθερές 128-bit **c1, c2, c3, c4** και **c5** ο οποίες ορίζονται ως εξής:

$$c1[i] = 0 \text{ for } 0 \leq i \leq 127$$
$$c2[i] = 0 \text{ for } 0 \leq i \leq 127, \text{ εκτός από το } c2[127] = 1$$
$$c3[i] = 0 \text{ for } 0 \leq i \leq 127, \text{ εκτός από το } c3[126] = 1$$
$$c4[i] = 0 \text{ for } 0 \leq i \leq 127, \text{ εκτός από το } c4[125] = 1$$
$$c5[i] = 0 \text{ for } 0 \leq i \leq 127 \text{ εκτός από το } c5[124] = 1$$

Πέντε ακέραιοι **r1, r2, r3, r4** και **r5** ο οποίοι ορίζονται ως εξής:

$$r1 = 64; r2 = 0; r3 = 32; r4 = 64; r5 = 96$$

Πέντε τιμές εξόδου **OUT1, OUT2, OUT3, OUT4** και **OUT5** οι οποίες υπολογίζονται με τον παρακάτω τρόπο.

OUT1= E[TEMP⊕rot(IN1⊕OPc, r1) ⊕c1]K⊕OPc
OUT2= E[rot(TEMP⊕OPc, r2) ⊕c2]K⊕OPc
OUT3= E[rot(TEMP⊕OPc, r3) ⊕c3]K⊕OPc
OUT4= E[rot(TEMP⊕OPc, r4) ⊕c4]K⊕OPc
OUT5= E[rot(TEMP⊕OPc, r5) ⊕c5]K⊕OPc

Οι έξοδοι των αλγορίθμων **f1, f1*, f2, f3, f4, f5** και **f5*** ορίζονται ως:

Έξοδος του **f1**= MAC-A,
όπου MAC-A[0] .. MAC-A[63] = OUT1[0] .. OUT1[63]

Έξοδος του **f1***= MAC-S,
όπου MAC-S[0] .. MAC-S[63] = OUT1[64] .. OUT1[127]

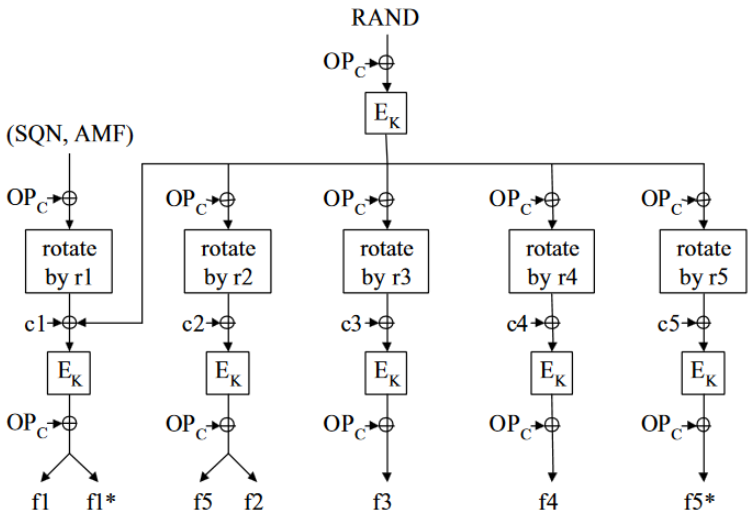
Έξοδος του **f2**= RES,
όπου RES[0] .. RES[63] = OUT2[64] .. OUT2[127]

Έξοδος του **f3**= CK,
όπου CK[0] .. CK[127] = OUT3[0] .. OUT3[127]

Έξοδος του **f4**= IK,
όπου IK[0] .. IK[127] = OUT4[0] .. OUT4[127]

Έξοδος του **f5**= AK,
όπου AK[0] .. AK[47] = OUT2[0] .. OUT2[47]

Έξοδος του **f5***= AK,
όπου AK[0] .. AK[47] = OUT5[0] .. OUT5[47]



Εικόνα 4.2 Ορισμός των αλγορίθμων του MILENAGE

4.4.1 Πλαίσιο Εκ - Rijndael

Ο τρόπος λειτουργίας του αλγορίθμου Εκ (Rijndael), κατά τον σχεδιασμό του αλγορίθμου, ορίσαμε το παρακάτω πλαίσιο.

ΠΛΑΙΣΙΟ ΑΛΓΟΡΙΘΜΟΥ Εκ

Ορίζεται το συστατικό **KeySchedule**, με είσοδο το key και έξοδο τα 11 round_keys(i), με την χρήση του σήματος tmp_roundkey και σταθερά το roundConst λειτουργώντας ως εξής:

roundConst = ("00000001", "00000010", "00000100", "00001000", "00010000", "00100000", "01000000", "10000000", "00011011","00110110")

Για i από 1 έως 10 έχουμε

tmp_roundkey(i) = S_box[tmp_roundkey](i) ⊕ tmp_roundkey(i) ⊕ roundConst(i)

roundkeys(i) = tmp_roundkeys(i)

Ορίζεται το συστατικό **KeyAdd** με εισόδους το plaintext/ shift_out(10) και το round_keys(i) και η έξοδος δημιουργείται ως εξής:

add_out(i) = plaintext/ shift_out(10) ⊕ round_keys(i)

Το συστατικό **ByteSub**, το οποίο δέχεται ως είσοδο το add_out και η έξοδος υπολογίζεται ως εξής:

sub_out(i) = S_box[add_out(i)]

Το συστατικό **ShiftRows**, ορίζεται με εισόδους το sub_out(i) και έξοδο την τιμή shift_out(i) η οποία έχει ολισθήσει.

Το συστατικό **MixColums** το οποίο δέχεται ως είσοδο το shift_out και ως εσωτερικά σήματα ορίζονται το mult_2, mult3, shift_2, shift3 και state δημιουργώντας την έξοδο με την εξής λειτουργία:

mix_out(i) = state(i) ⊕ mult_2(i) ⊕ mult_3(i)

Η έξοδος του Ek ορίζεται ως εξής:

cipher_text = KeyAdd[shift_out(10), round_keys],

με **shift_out(10)** = ShiftRow[sub_out(10)],

με **sub_out(10)** = ByteSub[add_out(10)]

ΚΕΦΑΛΑΙΟ 5
ΑΝΑΠΤΥΞΗ ΚΩΔΙΚΑ VHDL

5.1 Υλοποίηση Μηχανισμού Ασφαλείας MILENAGE

Κατά την υλοποίηση της παρούσας πτυχιακής εργασίας η γλώσσα σχεδιασμού υλικού η οποία χρησιμοποιήθηκε για την ανάπτυξη του κώδικα περιγραφής του μηχανισμού ασφαλείας MILENAGE είναι η VHDL καθώς αποτελεί την δημοφιλέστερη γλώσσα σχεδιασμού υλικού.

Αρχικά υλοποιούμε την ανάπτυξη του κώδικα περιγραφής υλικού και αφού ελέγξουμε με την λειτουργία της μετάφρασης την συντακτική ορθότητα του κώδικα στην συνέχεια γίνεται εξομοίωση του με την χρήση των dofiles για τον έλεγχο της λογικής ορθότητας.

5.2 Γλώσσα Σχεδιασμού Υλικού VHDL

Η VHDL είναι η δημοφιλέστερη γλώσσα σχεδιασμού υλικού, hardware, και χρησιμοποιείται για την ανάπτυξη ολοκληρωμένων ψηφιακών ηλεκτρονικών κυκλωμάτων και συστημάτων. Ο όρος VHDL είναι συντόμευση των λέξεων VHSIC Hardware Description Language, όπου VHSIC σημαίνει Very High Speed Integrated Circuit.

Η γλώσσα VHDL χρησιμοποιείται στον ηλεκτρονικό σχεδιασμό για την περιγραφή της συμπεριφοράς, την εφαρμογή και την δομή των ψηφιακών συστημάτων. Βοηθάει τους σχεδιαστές να κάνουν μια υψηλού επιπέδου σχεδίαση και έπειτα με την χρήση των εργαλείων σύνθεσης υλοποιούν την αποτύπωση σε ολοκληρωμένη μορφή βάση τον προδιαγραφών οι οποίο έχουν ορισθεί κατά τον σχεδιασμό.

Η VHDL αποτελεί μία από τις πιο σημαντικές γλώσσες περιγραφής υλικού για να καθορίσει, να επιβεβαιώσει και να σχεδιάσει ηλεκτρονικά κυκλώματα. Το βασικό πλεονέκτημα της είναι ότι επιτρέπει την περιγραφή και τον έλεγχο ορθής λειτουργία του συστήματος το οποίο έχει σχεδιαστεί, πριν τα εργαλεία σύνθεσης μεταφράσουν τη σχεδίαση σε πραγματικό υλικό, δηλαδή πύλες και γραμμές. Επίσης υποστηρίζει συντρέχουσες και ακολουθιακές δομές ο οποίες είναι απαραίτητες για την περιγραφή της λειτουργίας υλικού και παρέχει εύκολη διαχείριση λαθών και επιβεβαίωση ορθής λειτουργίας μέσω των λειτουργιών Simulation, Error Management και Design Verification.

Υποστηρίζει ιεραρχική σχεδίαση, δηλαδή μπλοκ και επαναχρησιμοποιούμενα συστατικά. Το κάθε μπλοκ μπορεί να περιγραφεί μέσω διαδικασιών και συναρτήσεων. Βασική ιδέα είναι η απόκρυψη των μη χρήσιμων πληροφοριών σε κάθε ιεραρχικό επίπεδο και υποστηρίζεται η αρχή του μαύρου κουτιού, δηλαδή μόνο οι είσοδοι, οι έξοδοι και η λειτουργία του κάθε συστατικού είναι ορατά.

5.2.1 Πεδία Εφαρμογής VHDL

Οι λειτουργίες στις οποίες μπορούμε να χρησιμοποιήσουμε την VHDL είναι οι παρακάτω:

- Εξομοίωση ορθής λειτουργίας (Simulation).
- Σύνθεση ψηφιακών κυκλωμάτων (Synthesis).
- Επιβεβαίωση ορθού σχεδιασμού(Design Verification).
- Μοντέλα προδιαγραφών(Specification Models).

5.2.2 Χαρακτηριστικά VHDL

Τα χαρακτηριστικά τα οποία διακρίνουν την VHDL παρουσιάζονται συνοπτικά στην συνέχεια:

- Όπως και σε ένα οποιοδήποτε ψηφιακό σύστημα έτσι και με την VHDL μπορούμε να έχουμε ταυτόχρονη δήλωση λειτουργίας των στοιχείων από τα οποία αποτελείται το σύστημα. Με αυτόν τον τρόπο είναι δυνατή η ταυτόχρονη λειτουργία τους.
- Για να είναι δυνατή η διαδοχική εκτέλεση των λειτουργιών οι οποίες ορίζονται θα πρέπει να παρέχεται σειριακή δήλωση λειτουργίας.
- Παρέχεται η υποστήριξη βιβλιοθηκών με σκοπό να χρησιμοποιούνται τύποι δεδομένων που έχουν ορισθεί από τους χρήστες και με την βοήθεια του compiler να μεταφράζονται.
- Παρέχεται η υποστήριξη της ιεραρχικής σχεδίασης.
- Προκειμένου να καθορίζονται ευκολότερα οι λειτουργίες και η συναρτήσεις γίνεται η χρήση υποπρογραμμάτων.

5.2.3 Συντρέχων και Ακολουθιακός Κώδικας

Από τον σχεδιασμό της, η VHDL αποτελεί έναν συντρέχων κώδικα, δηλαδή οι λειτουργίες του εκτελούνται παράλληλα. Μόνο μέσα σε μία Διεργασία – Process, Συνάρτηση – Function και Διαδικασία - Procedure ο κώδικας εκτελείται ακολουθιακά δηλαδή σειριακά. Στις παραπάνω λειτουργίες η εκτέλεση του κώδικα είναι επίσης ακολουθιακή αλλά σε σχέση με τον υπόλοιπο κώδικα εκτελείται παράλληλα.

Με την χρήση ενός συντρέχων κώδικα μπορούμε να υλοποιήσουμε μόνο συνδυαστικά κυκλώματα ενώ με την χρήση ενός ακολουθιακού κώδικα μπορούμε να υλοποιήσουμε και ακολουθιακά και συνδυαστικά κυκλώματα.

Συνδυαστικά Κυκλώματα, είναι τα κυκλώματα όπου η έξοδός τους εξαρτάται αποκλειστικά από τις τρέχουσες εισόδους. Για αυτό τον λόγο δεν είναι αναγκαία η χρήσης μνήμης και συνεπώς μπορούν να υλοποιηθούν με την χρήση συμβατικών πυλών.

Ακολουθιακά Κυκλώματα, είναι τα κυκλώματα όπου η έξοδός τους εξαρτάται και από προηγούμενες εισόδους. Για αυτό τον λόγο είναι αναγκαία η χρήση στοιχείων μνήμης, στις οποίες αποθηκεύονται οι καταστάσεις που επηρεάζουν τις εξόδους του κυκλώματος.

5.2.4 Δομικά Στοιχεία Σχεδιασμού

Με την ανάπτυξη κώδικα με χρήση VHDL περιγράφεται ο τρόπος διασύνδεσης ενός κυκλώματος με το εξωτερικό περιβάλλον και η εσωτερική υλοποίηση του. Ο σχεδιασμός αποτελείται από τις οντότητες (entities) και τις αρχιτεκτονικές (architectures). Στα ολοκληρωμένα κυκλώματα υπάρχει η δυνατότητα χρήσης πολλών οντοτήτων και αρχιτεκτονικών όπως επίσης υπάρχει και η δυνατότητα να δημιουργίας ενός ιεραρχικού κυκλώματος. Δηλαδή περιέχεται μία οντότητα μέσα στην άλλη χρησιμοποιώντας υποκυκλώματα τα οποία ονομάζονται συστατικά (components).

Σε κάθε επίπεδο της ροής σχεδιασμού μπορεί να χρησιμοποιηθούν τρεις μέθοδοι περιγραφής κυκλωμάτων

- Συμπεριφοράς (Behavioral), χρησιμοποιείται για την μοντελοποίηση της συμπεριφοράς του κυκλώματος σε υψηλό και αφηρημένο επίπεδο. Πιο συγκεκριμένα περιγράφεται αλγοριθμικά η λειτουργία του κυκλώματος, δεν γίνεται αναλυτική περιγραφή της δομής του κυκλώματος και δεν απαιτούνται αναλυτικές λογικές εξισώσεις.
- Ροής Δεδομένων (Data Flow), χρησιμοποιείται για την αναλυτικότερη περιγραφή της λειτουργίας του κυκλώματος, γίνεται χρήση λογικών εξισώσεων για την μοντελοποίηση της ροής δεδομένων και δίνεται η δυνατότητα της εξομοίωσης και της επιβεβαίωσης ορθής λειτουργίας.
- Δομής (Structural), χρησιμοποιείται για την περιγραφή της διασύνδεσης των δομικών μονάδων του κυκλώματος με την προϋπόθεση της ύπαρξης βιβλιοθήκης από σχεδιασθέντα δομικά στοιχεία και την δυνατότητα χρήσης τους.

5.3 Ορισμοί και Τύποι Δεδομένων

Για την υλοποίηση του μηχανισμού ασφαλείας της παρούσας πτυχιακής εργασίας με την χρήση της γλώσσας περιγραφής υλικού, VHDL, ορίσαμε τις εξής διαφοροποιήσεις σε σχέση με τον αρχικό σχεδιασμό.

1. Κατά την ανάπτυξη του κώδικα με την χρήση της VHDL, οι πίνακες που χρησιμοποιούνται ως Κατάσταση (State) για τον σχεδιασμό του αλγορίθμου Rijndael είναι μονοδιάστατοι [16] byte, αντί για δισδιάστατους [4x4] byte που χρησιμοποιήθηκαν κατά τον σχεδιασμό.
2. Η μετατροπή της 128-bit εισόδου του αλγορίθμου Rijndael καθώς και όλων των λειτουργιών του σε μονοδιάστατο πίνακα των 16 byte γίνεται εντός των

αντίστοιχων λειτουργιών. Με τον ίδιο τρόπο τα αποτελέσματα των λειτουργιών μετατρέπονται σε σήμα 128bit.

3. Χρησιμοποιούμε τιμή για το Πεδίο Διαμόρφωσης του Αλγορίθμου (OP) την οποία έχει επιλέξει και ορίσει ο διαχειριστής χωρίς να είναι δημόσια. Η τιμή της OP δηλαδή έχει ορισθεί σαν σταθερά στους κώδικες των αλγορίθμων και όχι σαν είσοδος των αλγορίθμων.

Ο ορισμός πακέτων (packages) χρησιμοποιείται για την αποθήκευση ενός συνόλου τύπων δεδομένων, διαδικασιών, συναρτήσεων και κοινών στοιχείων τα οποία χρησιμοποιούνται συχνά από τους σχεδιαστές. Με αυτόν τον τρόπο γίνεται πιο απλή και ευέλικτη η ανάπτυξη του κώδικα.

Οι παρακάτω τύποι δεδομένων έχουν ορισθεί στο πακέτο my_package και η περιγραφή του δίνεται στο Παράρτημα Ι.

- **row:** μονοδιάστατος πίνακας 16 κελιών όπου το κάθε κελί αποτελείται από σήμα 8-bit.
- **matrix:** μονοδιάστατος πίνακας 11 κελιών όπου το κάθε κελί αποτελείται από σήμα τύπου row.
- **key_matrix**: μονοδιάστατος πίνακας 11 κελιών όπου το κάθε κελί αποτελείται από σήμα 128-bit.
- **key_matrix2**: μονοδιάστατος πίνακας 10 κελιών όπου το κάθε κελί αποτελείται από σήμα 128-bit.
- **key_matrix3**: μονοδιάστατος πίνακας 9 κελιών όπου το κάθε κελί αποτελείται από σήμα 128-bit.
- **round_array**: μονοδιάστατος πίνακας 10 κελιών όπου το κάθε κελί αποτελείται από σήμα 8-bit.
- **mix_array:** μονοδιάστατος πίνακας 16 κελιών όπου το κάθε κελί αποτελείται από σήμα 9-bit.

5.4 Ανάπτυξη Κώδικα MILENAGE με χρήση VHDL

Στην συγκεκριμένο κεφάλαιο βλέπουμε την ανάλυση του κώδικα, ο οποίος περιγράφει την λειτουργία του MILENAGE. Η σύνταξη του κώδικα έχει υλοποιηθεί με την χρήση της γλώσσας VHDL και δίνεται στο Παράρτημα Ι.

5.4.1 Κώδικας MILENAGE

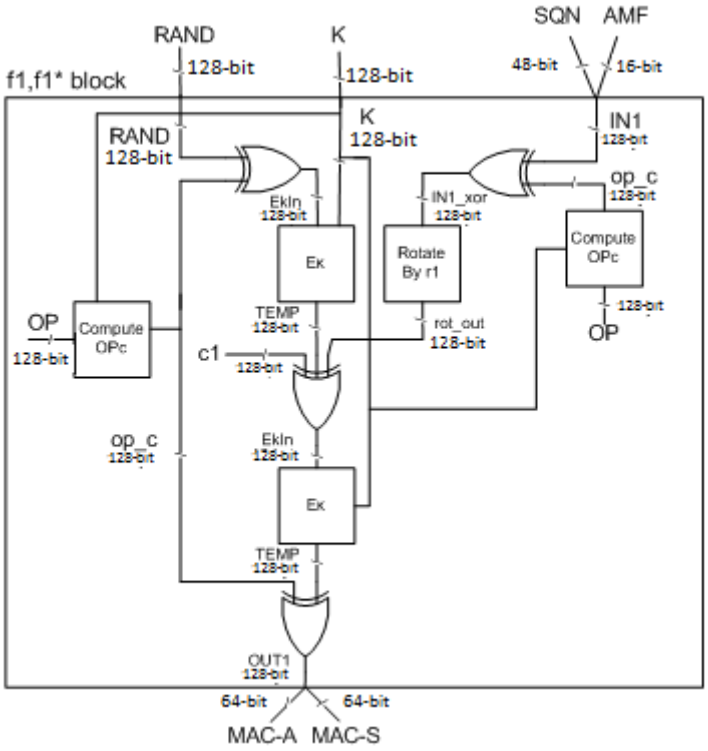
Στον κώδικα του MILENAGE αρχικά ορίζονται οι είσοδοι του, οι οποίοι είναι οι εξής, το 128-bit K, το 128-bit RAND, το 48-bit SQN και το 16-bit AMF. Στην συνέχεια ορίζονται τα συστατικά (components) των αλγορίθμων f1, f1star, f2, f3, f4, f5 και f5star με τις αντίστοιχες εισόδους και εξόδους καθώς και τα εσωτερικά σήματα TEMP1, TEMP2, TEMP3, TEMP4 και TEMP5, 128-bit το καθένα.

Με την χρήση του δομικού (structural) κώδικα γίνεται αντιστοιχία των εισόδων-εξόδων του κάθε component των f1, f1star, f2, f3, f4, f5 και f5star με τα εσωτερικά

σήματα TEMP1, TEMP2, TEMP3, TEMP4 και TEMP5. Τέλος τα εσωτερικά σήματα μετατρέπονται στις εξόδους του MILENAGE, δηλαδή τα 128-bit f1_out, f1star_out, f2_out, f3_out, f4_out, f5_out και f5star_out.

5.4.2. Κώδικας f1 και f1star

Στους κώδικες f1 και f1star ορίζονται οι είσοδοι των αλγορίθμων f1 και f1star, το 128-bit K, το 128-bit RAND, το 48-bit SQN και το 16-bit AMF και στην συνέχεια ορίζονται τα συστατικά (components) του αλγορίθμου RijndaelEncrypt και της διαδικασίας Compute_opc. Τα εσωτερικά σήματα τα οποία χρησιμοποιούνται είναι τα εξής: OP_c, ek_in1, IN1, IN1_xor, TEMP1, rot_out, ek_in2 και TEMP2, όπως επίσης οι σταθερές c1 και OP, 128-bit το καθένα.



Εικόνα 5.1 Λειτουργία αλγορίθμων f1 και f1*

Καλείται η διαδικασία Compute_opc με εισόδους το OP και το K και έξοδο το εσωτερικό σήμα OP_c. Αφού το OP_c γίνει XORed με την είσοδο RAND, η έξοδος ek_in1 που παράγεται, χρησιμοποιείται ως είσοδος μαζί με το K στο component RijndaelEncrypt με έξοδο το TEMP1. Στην συνέχεια δημιουργείται το σήμα IN1 με την ένωση των εισόδων SQN και AMF το οποίο γίνεται XORed με το σήμα OP_c και παράγεται το IN1_xor. Αφού γίνει ολίσθηση του σήματος IN1_xor κατά 64 bits και λάβουμε το σήμα rot_out, στην συνέχεια γίνεται η πράξη XOR στα σήματα c1, TEMP1 και rot_out παράγοντας το ek_in2.

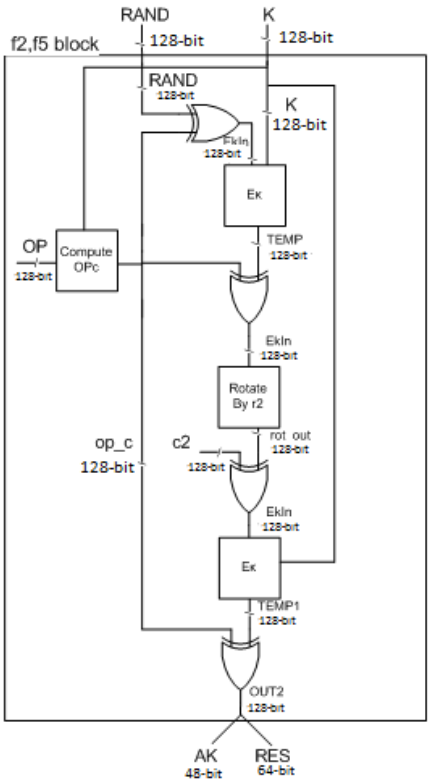
Τέλος, καλείται ξανά το συστατικό RijndaelEncrypt με εισόδους το ek_in2 και το K και λαμβάνουμε το σήμα TEMP2. Η έξοδος των αλγορίθμων f1 και f1star, OUT1 δημιουργείται με την πράξη XOR ανάμεσα στα σήματα OP_c και TEMP2.

5.4.3 Κώδικας f2 και f5

Αρχικά ορίζονται οι είσοδοι των αλγορίθμων f2 και f5, το 128-bit K και το 128-bit RAND και στην συνέχεια ορίζονται τα συστατικά (components) του αλγορίθμου RijndaelEncrypt και της διαδικασίας Compute_opc. Τα εσωτερικά σήματα τα οποία χρησιμοποιούνται είναι τα εξής: OP_c, ek_in, TEMP1, rot_in, rot_out, ek_in2 και TEMP2, όπως επίσης οι σταθερές c2 και OP, 128-bit το καθένα.

Καλείται η διαδικασία Compute_opc με εισόδους το OP και το K και έξοδο το εσωτερικό σήμα OP_c. Αφού το OP_c γίνει XORed με την είσοδο RAND, η έξοδος ek_in1 που παράγεται, χρησιμοποιείται ως είσοδος μαζί με το K στο component RijndaelEncrypt με έξοδο το TEMP1. Αφού η έξοδος TEMP1 γίνει XORed με το σήμα OP_c παράγεται σήμα το rot_in ίσο. Το σήμα rot_out είναι ίσο με το rot_in καθώς δεν γίνεται κάποια μετατόπιση. Στην συνέχεια γίνεται η πράξη XOR στα σήματα c2 και rot_out παράγοντας το ek_in2.

Τέλος, καλείται ξανά το συστατικό RijndaelEncrypt με εισόδους το ek_in2 και το K και λαμβάνουμε το σήμα TEMP2. Η έξοδος των αλγορίθμων f2 και f5, OUT2 128-bit, δημιουργείται με την πράξη XOR ανάμεσα στα σήματα OP_c και TEMP2.



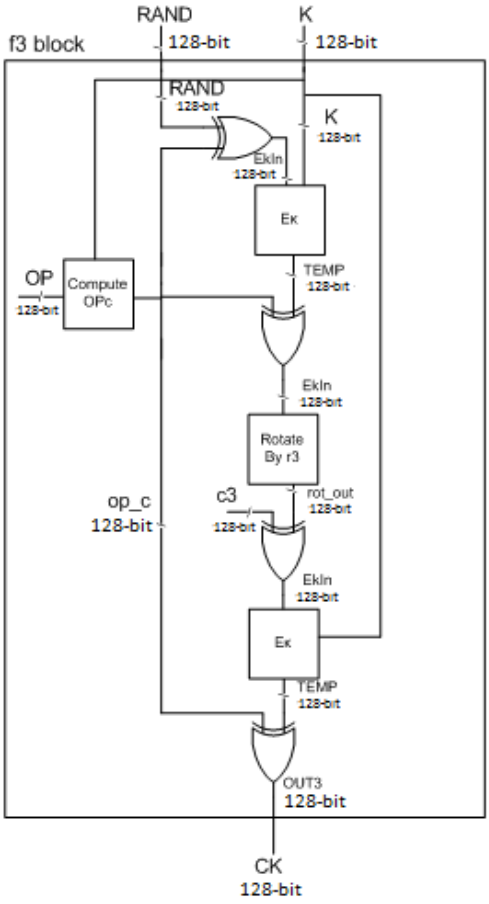
Εικόνα 5.2 Λειτουργία αλγορίθμων f2 και f5

5.4.4 Κώδικας f3

Στην αρχή ορίζονται οι είσοδοι του αλγορίθμου f3, το 128-bit K και το 128-bit RAND και στην συνέχεια ορίζονται τα συστατικά του αλγορίθμου RijndaelEncrypt και της διαδικασίας Compute_opc. Τα εσωτερικά σήματα τα οποία χρησιμοποιούνται είναι τα εξής: OP_c, ek_in1, TEMP1, rot_out, ek_in2 και TEMP2, όπως επίσης οι σταθερές c3 και OP, 128-bit το καθένα.

Καλείται η διαδικασία Compute_opc με εισόδους το OP και το K και έξοδο το εσωτερικό σήμα OP_c. Αφού το OP_c γίνει XORed με την είσοδο RAND, η έξοδος ek_in1 που παράγεται, χρησιμοποιείται ως είσοδος μαζί με το K στο component RijndaelEncrypt με έξοδο το TEMP1. Στην συνέχεια δημιουργείται το σήμα rot_in με την πράξη XOR ανάμεσα στα σήματα TEMP1 και OP_c. Αφού γίνει ολίσθηση του σήματος rot_in κατά 32 bits και λάβουμε το σήμα rot_out, στην συνέχεια γίνεται η πράξη XOR στα σήματα c3 και rot_out παράγοντας το ek_in2.

Τέλος, καλείται εκ νέου το συστατικό RijndaelEncrypt με εισόδους το ek_in2 και το K και λαμβάνουμε το σήμα TEMP2. Η έξοδος του αλγορίθμου f3, OUT3 128-bit, δημιουργείται με την πράξη XOR ανάμεσα στα σήματα OP_c και TEMP2.



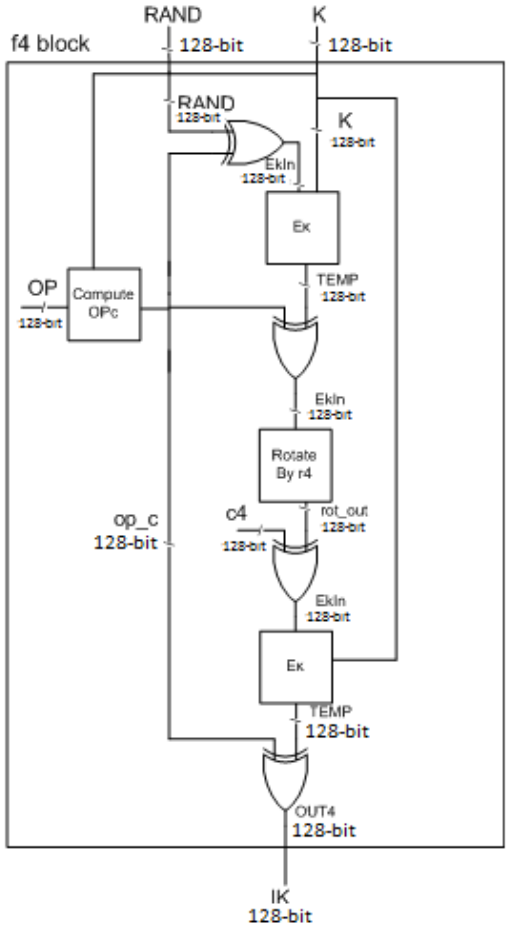
Εικόνα 5.3 Λειτουργία αλγορίθμου f3

5.4.5 Κώδικας f4

Αρχικά ορίζονται οι είσοδοι του αλγορίθμου f4, το 128-bit K και το 128-bit RAND και στην συνέχεια ορίζονται τα συστατικά του αλγορίθμου RijndaelEncrypt και της διαδικασίας Compute_opc. Τα εσωτερικά σήματα τα οποία χρησιμοποιούνται είναι τα εξής: OP_c, ek_in1, TEMP1, rot_out, ek_in2 και TEMP2, όπως επίσης οι σταθερές c4 και OP, 128-bit το καθένα.

Καλείται η διαδικασία Compute_opc με εισόδους το OP και το K και έξοδο το εσωτερικό σήμα OP_c. Αφού το OP_c γίνει XORed με την είσοδο RAND, η έξοδος ek_in1 που παράγεται, χρησιμοποιείται ως είσοδος μαζί με το K στο component RijndaelEncrypt με έξοδο το TEMP1. Στην συνέχεια δημιουργείται το σήμα rot_in με την πράξη XOR ανάμεσα στα σήματα TEMP1 και OP_c. Αφού γίνει ολίσθηση του σήματος rot_in κατά 64 bits και λάβουμε το σήμα rot_out, στην συνέχεια γίνεται η πράξη XOR στα σήματα c4 και rot_out παράγοντας το ek_in2.

Τέλος, καλείται εκ νέου το συστατικό RijndaelEncrypt με εισόδους το ek_in2 και το K και λαμβάνουμε το σήμα TEMP2. Η έξοδος του αλγορίθμου f4, OUT4 128-bit, δημιουργείται με την πράξη XOR ανάμεσα στα σήματα OP_c και TEMP2.



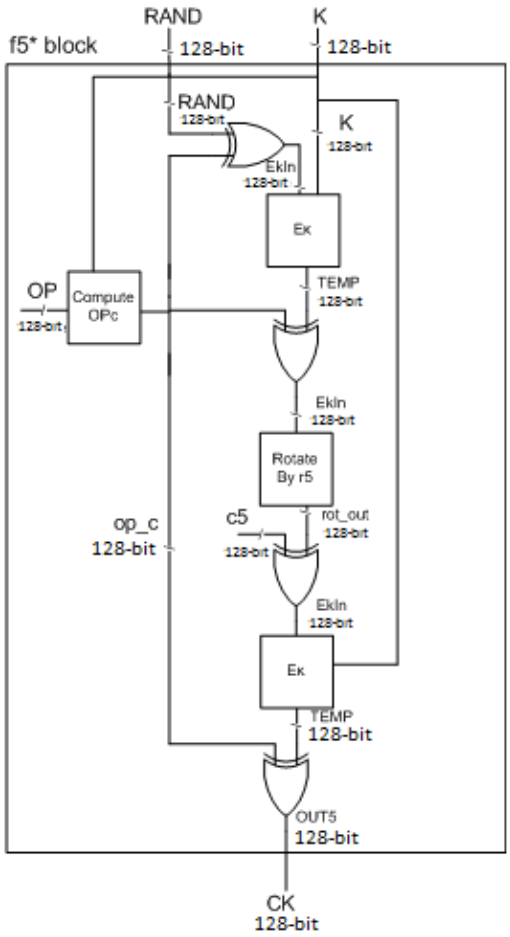
Εικόνα 5.4 Λειτουργία αλγορίθμου f4

5.4.6 Κώδικας f5star

Αρχικά ορίζονται οι είσοδοι του αλγορίθμου f5star, το 128-bit K και το 128-bit RAND και στην συνέχεια ορίζονται τα συστατικά του αλγορίθμου RijndaelEncrypt και της διαδικασίας Compute_opc. Τα εσωτερικά σήματα τα οποία χρησιμοποιούνται είναι τα εξής: OP_c, ek_in1, TEMP1, rot_out, ek_in2 και TEMP2, όπως επίσης οι σταθερές c5 και OP, 128-bit το καθένα.

Καλείται η διαδικασία Compute_opc με εισόδους το OP και το K και έξοδο το εσωτερικό σήμα OP_c. Αφού το OP_c γίνει XORed με την είσοδο RAND, η έξοδος ek_in1 που παράγεται, χρησιμοποιείται ως είσοδος μαζί με το K στο component RijndaelEncrypt με έξοδο το TEMP1. Στην συνέχεια δημιουργείται το σήμα rot_in με την πράξη XOR ανάμεσα στα σήματα TEMP1 και OP_c. Αφού γίνει ολίσθηση του σήματος rot_in κατά 96 bits και λάβουμε το σήμα rot_out, στην συνέχεια γίνεται η πράξη XOR στα σήματα c5 και rot_out παράγοντας το ek_in2.

Τέλος, καλείται εκ νέου το συστατικό RijndaelEncrypt με εισόδους το ek_in2 και το K και λαμβάνουμε το σήμα TEMP2. Η έξοδος του αλγορίθμου f5, OUT5 128-bit, δημιουργείται με την πράξη XOR ανάμεσα στα σήματα OP_c και TEMP2.



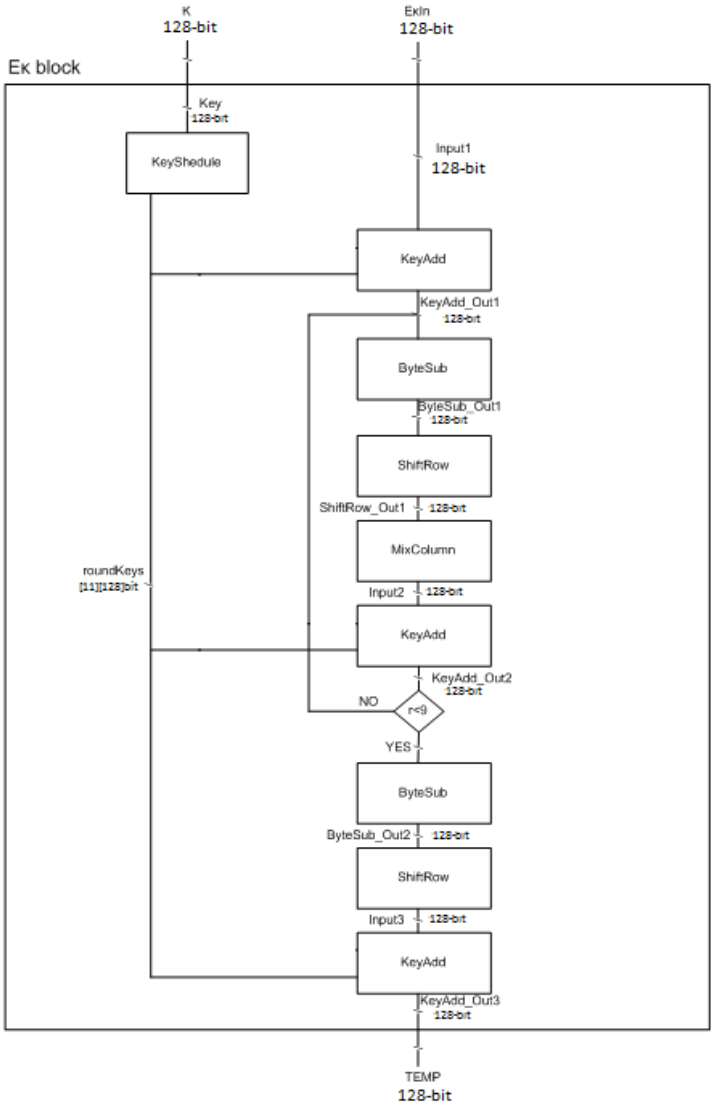
Εικόνα 5.5 Λειτουργία αλγορίθμου f5

5.4.7 Κώδικας RijndaelEncrypt (Εκ)

Οι είσοδοι του αλγορίθμου Ek – RijdaelEncrypt είναι τα 128-bit plaintext και key και τα συστατικά τα οποία αποτελούν την δομή του αλγορίθμου είναι τα εξής. Τα KeySchedule, KeyAdd, ByteSub, ShiftRow και MixColumn. Τα εσωτερικά σήματα ορίζοντα τα round_keys, sub_out, mix_out τύπου key_matrix, key_matrix2 και key_matrix3 αντίστοιχα.

Αρχικά καλείται η διαδικασία KeySchedule με είσοδο το key και έξοδο το roundkeys και με την σειρά καλείται το συστατικό KeyAdd με εισόδους το plaintext και το roundkey(0) και έξοδο το add_out(1). Έπειτα καλούνται τα συστατικά ByteSub, ShiftRow, MixColumn και KeyAdd με την συγκεκριμένη σειρά, για 9 επαναλαμβανόμενες φορές. Στο ByteSub έχουμε σαν είσοδο το add_out(i) και έξοδο το sud_out(i), στο ShiftRow έχουμε σαν είσοδο το sub_out(i) και έξοδο το shift_out(i), στο MixColumn έχουμε σαν είσοδο το shift_out(i) και έξοδο το mix_out(i) και στο KeyAdd έχουμε σαν εισόδους το mix_out(i) και το round_keys(i) και έξοδο το add_out(i+1).

Μετά τις 9 επαναλήψεις καλείται το συστατικό ByteSub με είσοδο το add_out(10) και έξοδο sub_out(10), μετά το συστατικό ShiftRow με είσοδο το sub_out(10) και έξοδο shift_out(10) και τέλος το KeyAdd συστατικά με εισόδους το shift_out(10) και το round_keys(10). Η έξοδος του τελευταίου συστατικού KeyAdd αποτελεί την 128-bit έξοδο του αλγορίθμου RijndaelEncrypt.

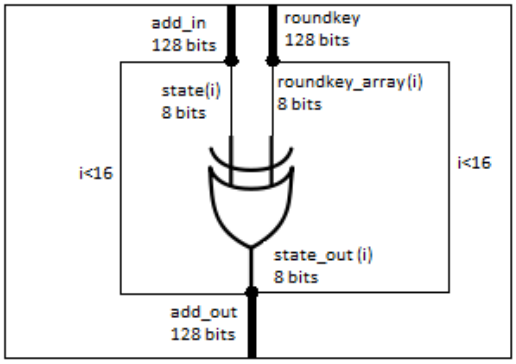


Εικόνα 5.6 Λειτουργία αλγορίθμου RijndaelEncrypt (Εκ)

5.4.8 Κώδικας KeyAdd

Σαν είσοδος στο συστατικό KeyAdd ορίζεται το 128-bit add_in και το roundkey 128-bit. Τα εσωτερικά σήματα είναι το state, το state_out και το roundkey_array τύπου row, δηλαδή δισδιάστατο πίνακα 16 x 128 bits.

Τα δύο σήματα εισόδου μετατρέπονται, μέσω της εντολής for…generate, σε δύο πίνακες 16 κελιών των 8 bytes το καθένα, με το add_in να μετατρέπεται σε state και το roundkey σε roundkey_array. Στην συνέχεια γίνεται XOR μεταξύ των δύο σημάτων state και roundkey_array και παράγεται ο πίνακας των 16 byte state_out. Τέλος, δημιουργείτε το σήμα εξόδου 128-bit add_out από το σήμα state_out μέσω μιας ακόμη for…generate.

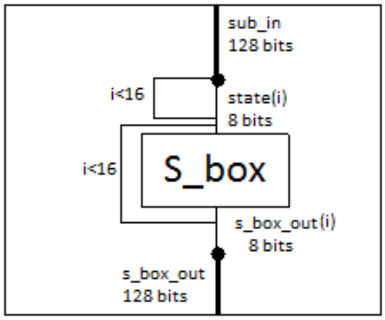


Εικόνα 5.7 Λειτουργία αλγορίθμου KeyAdd

5.4.9 Κώδικας ByteSub

Σαν είσοδος στο συστατικό ByteSub ορίζεται το 128-bit sub_in, το εσωτερικό σήμα state τύπου row και το συστατικό S_box.

Αρχικά η 128-bit είσοδος sub_in μετατρέπεται σε έναν μονοδιάστατο πίνακα (row) χωρίζοντας το διάνυσμα σε 16 κελιά των 8-bit παράγοντας την έξοδο state(i) εκτελώντας την εντολή for…generate. Στην συνέχεια, καλώντας το συστατικό S-box, γίνεται αντικατάσταση κάθε κελιού του πίνακα με το αντίστοιχο κελί του πίνακα S-box. Τέλος, δημιουργείτε ξανά ένα 128-bit σήμα το οποίο αποτελεί την έξοδο του αλγορίθμου sub_out.



Εικόνα 5.8 Λειτουργία αλγορίθμου ByteSub

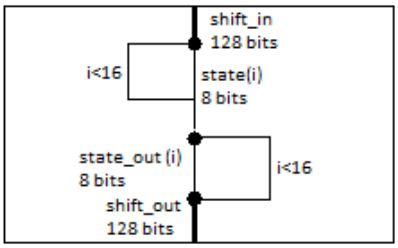
5.4.9.1 Κώδικας Sbox

Στο συστατικό S_box ορίζεται σαν είσοδος το 8-bit σήμα s_box_in. Για κάθε είσοδο αντιστοιχεί μία έξοδος 8-bit η οποία ορίζεται ως s_box_out.

5.4.10 Κώδικας ShiftRows

Σαν είσοδος στο συστατικό ShiftRows ορίζεται το 128-bit shift_in και τα εσωτερικά σήματα state και state_out τύπου row.

Αρχικά η 128-bit είσοδος shift_in μετατρέπεται μέσω της εντολής for…generate σε έναν μονοδιάστατο πίνακα (row) χωρίζοντας το διάνυσμα σε 16 κελιά των 8-bit παράγοντας την έξοδο state(i). Στην συνέχεια, γίνεται αντικατάσταση των κελιών έτσι ώστε να γίνουν οι απαιτούμενες ολισθήσεις και παράγεται ο πίνακας state_out(i). Τέλος, δημιουργείτε ξανά ένα 128-bit σήμα το οποίο αποτελεί την έξοδο του αλγορίθμου ShiftRows εκτελώντας την εντολή for…generate.



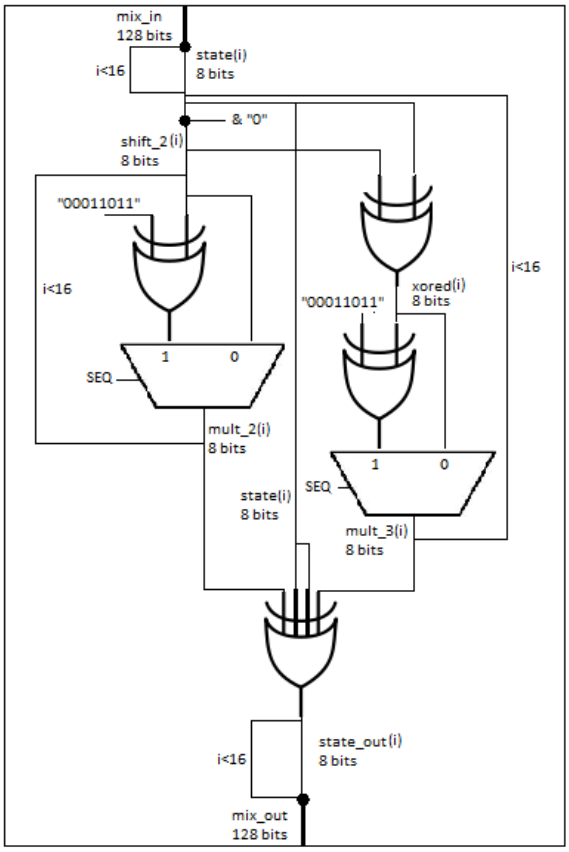
Εικόνα 5.9 Λειτουργία αλγορίθμου ShiftRows

5.4.11 Κώδικας MixColumns

Στο συστατικό MixColumns ορίζεται ως είσοδος το 128-bit mix_in. Τα εσωτερικά σήματα είναι το state και state_out, mult_2 και mult3 τύπου row όπως και τα σήματα shift_2, shift_3, xored τύπου mix_array.

Αρχικά η 128-bit είσοδος mix_in μετατρέπεται σε έναν μονοδιάστατο πίνακα (row) χωρίζοντας το διάνυσμα σε 16 κελιά των 8-bit παράγοντας σαν έξοδο τον πίνακα state(i) εκτελώντας την εντολή for…generate. Στην συνέχεια, εκτελείται η πράξη XOR ανάμεσα στο σήμα “00011011” και στο shift_2(i), το οποίο έχει παράγει η ένωση του σήματος state(i) με το ‘0’. Το σήμα που δημιουργείται μαζί με το σήμα shift_2(i) εισέρχεται σε έναν πολυπλέκτη 2in1 και ανάλογα με το SEQ παράγεται η έξοδος mult_2(i).

Το σήμα xored(i) παράγεται με την πράξη μεταξύ του σήματος shift_2(i) και του state(i). Το σήμα mult_3(i) παράγεται με την εκτέλεση της πράξης XOR μεταξύ του σήματος xored(i) και του ‘00011011’ όπου το παραγόμενο σήμα στην συνέχεια γίνεται είσοδος του πολυπλέκτη 2in1 μαζί με το σήμα xored(i). Τέλος, η εντολή XOR εκτελείται μεταξύ των σημάτων mult_2(i), mult_3(i) και state(i) παράγοντας το σήμα state_out(i). Η έξοδος mix_out παράγεται εκτελώντας την εντολή for…generate στο σήμα state_out(i).

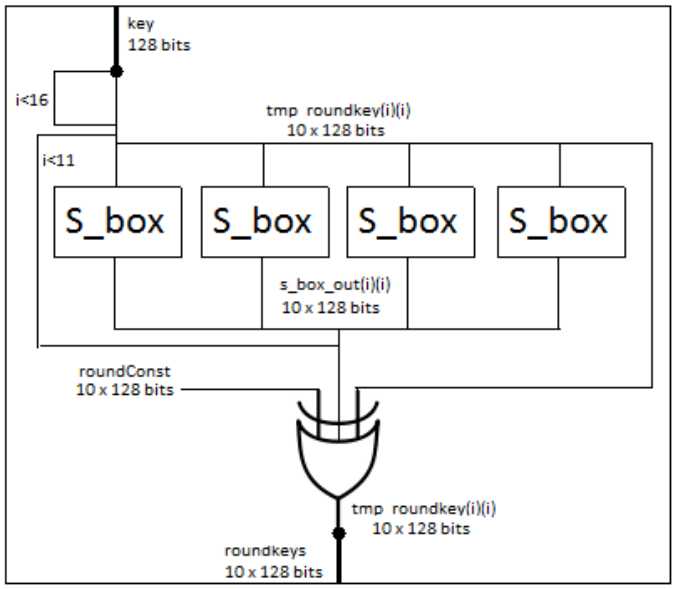


Εικόνα 5.10 Λειτουργία αλγορίθμου MixColumn

5.4.12 Κώδικας KeySchedule

Στο συστατικό KeySchedule ορίζεται ως είσοδος το 128-bit key και ως εσωτερικά σήματα ορίζετε το temp_roundkey τύπου matrix, η σταθερά roundConst όπως επίσης και 4 συστατικά S_box, με χρήση των σημάτων s_box_1, s_box_2, s_box_3, s_box_4 τύπου round_array.

Αρχικά η 128-bit είσοδος key μετατρέπεται σε έναν μονοδιάστατο πίνακα (row) χωρίζοντας το διάνυσμα σε 16 κελιά των 8-bit παράγοντας σαν έξοδο τον πίνακα tmp_round(i) εκτελώντας την εντολή for…generate. Στην συνέχεια, δημιουργούνται 4 στιγμιότυπα του S_box με είσοδο το tmp_round(i) και έξοδο του κάθε στιγμιότυπου τα σήματα s_box_1, s_box_2, s_box_3 και s_box_4. Τέλος, τα 11 κλειδιά roundkeys δημιουργούνται με την χρήση της XOR στα σήματα tmp_rounds(i), s_box_out(i) και roundConst.

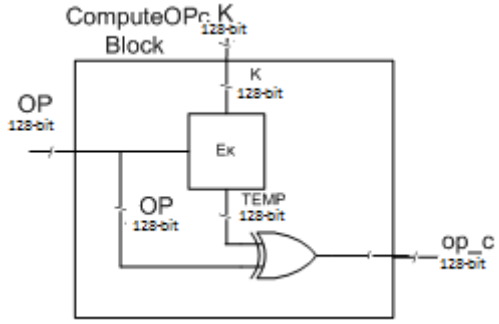


Εικόνα 5.11 Λειτουργία αλγορίθμου KeySchedule

5.4.13 Κώδικας Compute_opc

Σαν είσοδος στο συστατικό Compute_opc ορίζονται τα 128-bit σήματα OP και key, το εσωτερικό σήμα 128-bit, temp καθώς επίσης και η διαδικασία RijndaelEncrypt.

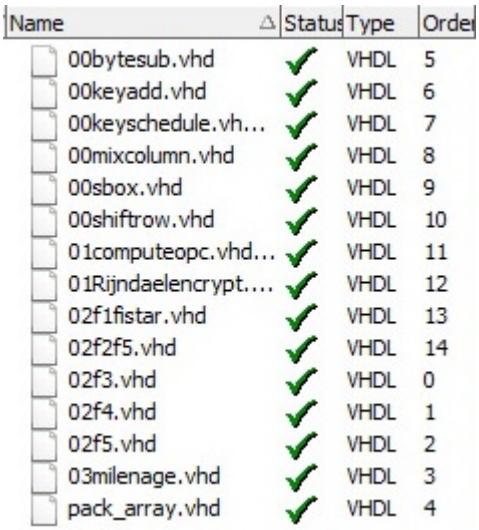
Αρχικά καλείται η διαδικασία RijndaelEncrypt με είσοδο τα σήματα OP και key και παράγεται το εσωτερικό σήμα temp. Στην συνέχεια το σήμα temp γίνεται XORed με το σήμα εισόδου OP όπου λαμβάνουμε την έξοδο του αλγορίθμου OP_c, 128-bit.



Εικόνα 5.12 Λειτουργία αλγορίθμου Compute_opc

5.5 Μετάφραση Κώδικα - Compilation

Με την ολοκλήρωση της περιγραφής του κυκλώματος ακολουθεί η διαδικασία της μετάφρασης (compilation). Ο μεταφραστής (compiler) αποτελείται από ένα σετ ανεξάρτητων εργαλείων που ελέγχουν και αναλύουν τον κώδικα VHDL για συντακτικά και λογικά λάθη. Στο κώδικα της παρούσας πτυχιακής έγινε έλεγχος της συντακτικής ορθότητας κάθε αλγορίθμου με το εργαλείο ModelSim PE Student Edition 10.4.a.




```
# Compile of 02f3.vhd was successful.
# Compile of 02f4.vhd was successful.
# Compile of 02f5.vhd was successful.
# Compile of 03milenage.vhd was successful.
# Compile of pack_array.vhd was successful.
# Compile of 00bytesub.vhd was successful.
# Compile of 00keyadd.vhd was successful.
# Compile of 00keyschedule.vhd was successful.
# Compile of 00mixcolumn.vhd was successful.
# Compile of 00sbox.vhd was successful.
# Compile of 00shiftrow.vhd was successful.
# Compile of 01computeopc.vhd was successful.
# Compile of 01Rijndaelencrypt.vhd was successful.
# Compile of 02f1fistar.vhd was successful.
# Compile of 02f2f5.vhd was successful.
# 15 compiles, 0 failed with no errors.
```

Εικόνα 5.13 Μετάφραση Κώδικα

Εφόσον ολοκληρωθεί επιτυχώς η διαδικασία της μετάφρασης του κάθε αλγόριθμου, όπως παρουσιάζεται και στην παραπάνω εικόνα, συνεχίζουμε την υλοποίηση του μηχανισμού με την επιβεβαίωση ορθής λειτουργίας.

5.6 Εξομοίωση - Επιβεβαίωση Ορθής Λειτουργίας

Η εξομοίωση των ολοκληρωμένων κυκλωμάτων αποτελεί μία σημαντική λειτουργία για την επιβεβαίωση της ορθής λειτουργίας του κώδικα κατά τον σχεδιασμό.

Στην παρούσα ενότητα παρουσιάζονται οι εξομοιώσεις που αποδεικνύουν τη σωστή λειτουργία του συστήματος MILENAGE που επιβεβαιώθηκε μετά από ενδελεχείς ελέγχους και διαδοχικές δοκιμές σύμφωνα με τα dofiles τα οποία δημιουργήθηκαν για το συγκεκριμένο σκοπό. Οι εντολές dofiles παρουσιάζονται στο Παράρτημα ΙΙ. Η εξομοίωση του υλοποιημένου συστήματος στην παρούσα πτυχιακή εργασία έγινε με χρήση του εργαλείου εξομοίωσης ModelSim PE Student Edition 10.4.a.

Τα dofiles αποτελούνται από εντολές οι οποίες εισάγουν τιμές σε όλες τις εισόδους των διαδικασιών (components) και επίσης συγχρονίζονται τα μηνύματα μεταξύ τους με την χρήση μιας συχνότητας. Μέσω του εργαλείου εξομοίωσης λαμβάνουμε τις τιμές των εξόδων και με αυτόν τον τρόπο ελέγχουμε την ορθή λειτουργία του μηχανισμού μας.

Για την ορθή λειτουργία του MILENAGE, δηλαδή των 7 αλγορίθμων (f1, f1*, f2, f3, f4, f5 και f5*) ορίσαμε μέσω του αρχείο dofiles τις παρακάτω τιμές για τις εισόδους σε δυαδικό σύστημα.

K: 01000110 01011011 01011100 11101000 10110001 10011001 10110100 10011111 10101010 01011111 00001010 00101110 11100010 00111000 10100110 10111100

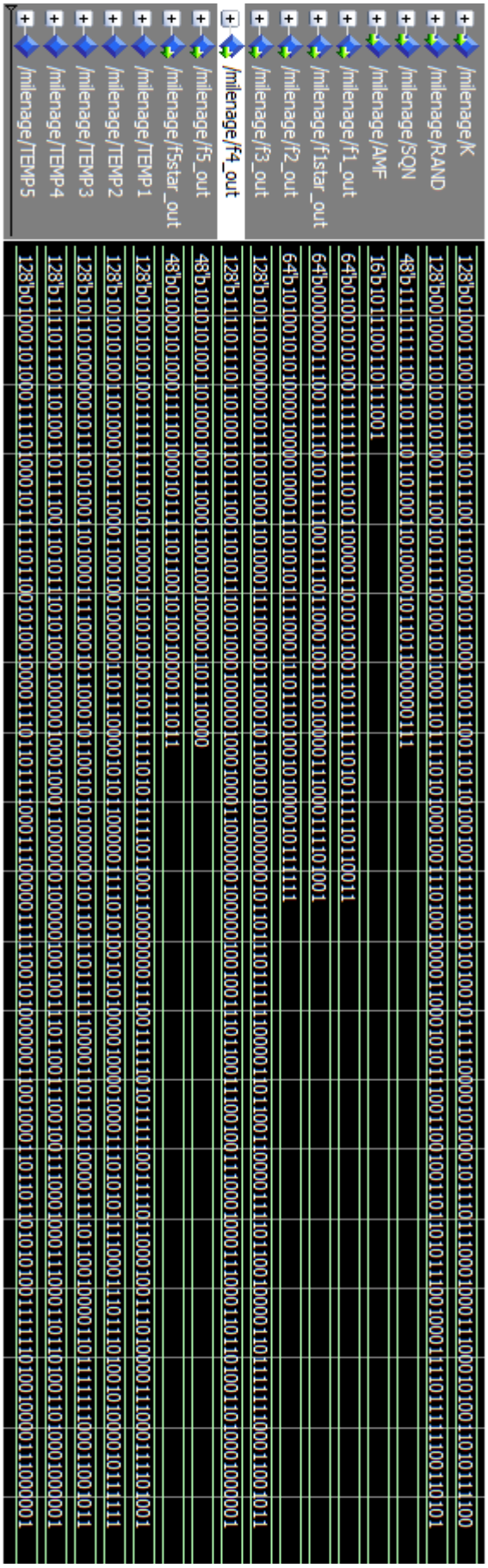
RAND: 00100011 01010101 00111100 10111110 10010110 00110111 10101000 10011101 00100001 10001010 11100110 01001101 10101110 01000111 10111111 00110101

SQN: 11111111 10011011 10110100 11010000 10110110 00000111

AMF: 10111001 10111001

OP: 11001101 11000010 00000010 11010101 00010010 00111110 00100000 11110110 00101011 01101101 01100111 01101010 11000111 00101100 10110011 00011000

Τα αποτελέσματα της εξομοίωσης παρουσιάζονται στην παρακάτω εικόνα σε δυαδικό σύστημα. Στα τέσσερα πρώτα σήματα αναπαριστούνται οι είσοδοι των αλγορίθμων υλοποίησης τις οποίες έχουμε ορίσει. Στην συνέχεια αναπαριστούνται τα σήματα TEMP, τα οποία αποτελούν ενδιάμεσα σήματα για την υλοποίηση των αλγορίθμων και τέλος εμφανίζονται τα αποτελέσματα του κάθε αλγορίθμου του MILENAGE



Εικόνα 5.14 Αποτελέσματα εξομοίωσης αλγορίθμων f1, f1*, f2, f3, f4, f5 και f5*

Τέλος βλέπουμε αναλυτικά τις εξόδους του κάθε αλγορίθμου.

f1_out: 01001010 10011111 11111010 11000011 01010100 11011111 10101111 10110011

f1*_out: 00000001 11001111 10101111 10011110 11000100 11101000 01110001 11101001

f2_out: 10100101 01000010 00010001 11010101 11100011 10111010 01010000 10111111

f5_out: 10101010 01101000 10011100 01100100 10000011 01110000

f3_out: 10110100 00001011 10101001 10100011 11000101 10001011 00101010 00000101 10111011 11110000 11011001 10000111 10110010 00011011 11111000 11001011

f4_out: 11110111 01101001 10111100 11010111 01010001 00000100 01000110 00000100 00010010 01110110 01110010 01110001 00011100 01101101 00110100 01000001

f5*_out: 01000101 00011110 10001011 11101100 10100100 00111011

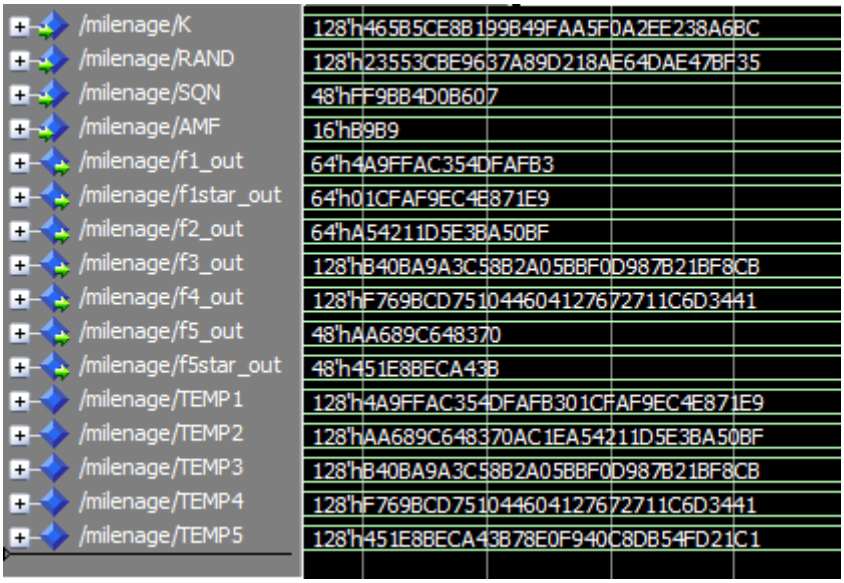
Για την ευκολότερη κατανόηση παρουσιάζονται στην συνέχεια οι είσοδοι, οι έξοδοι των αλγορίθμων καθώς και η εξομοίωση του μηχανισμού ασφαλείας σε δεκαεξαδικό σύστημα.

K: 465b5ce8 b199b49f aa5f0a2e e238a6bc

RAND: 23553cbe 9637a89d 218ae64d ae47bf35

SQN: ff9bb4d0 b607

AMF: b9b9



Εικόνα 5.15 Αποτελέσματα εξομοίωσης αλγορίθμων f1, f1*, f2, f3, f4, f5 και f5*

f1_out: 4a9ffac3 54dfafb3

f1*_out: 01cfaf9e c4e871e9

f2_out: a54211d5 e3ba50bf

f5_out: aa689c64 8370

f3_out: b40ba9a3 c58b2a05 bbf0d987 b21bf8cb

f4_out: f769bcd7 51044604 12767271 1c6d3441

f5*_out: 451e8bec a43b

5.7 Εξομοίωση - Επιβεβαίωση Ορθής Λειτουργίας 2

Για την επιβεβαίωση της ορθής λειτουργίας των αλγορίθμων του μηχανισμού ασφαλείας MILENAGE, με την χρήση των dofiles, ορίζουμε ένα δεύτερο σετ εισόδων των αλγορίθμων. Στην συνέχεια δίνεται το δεύτερο σετ εισόδων σε δεκαεξαδικό σύστημα.

K: 9e5944ae a94b8116 5c82fbf9 f32db751

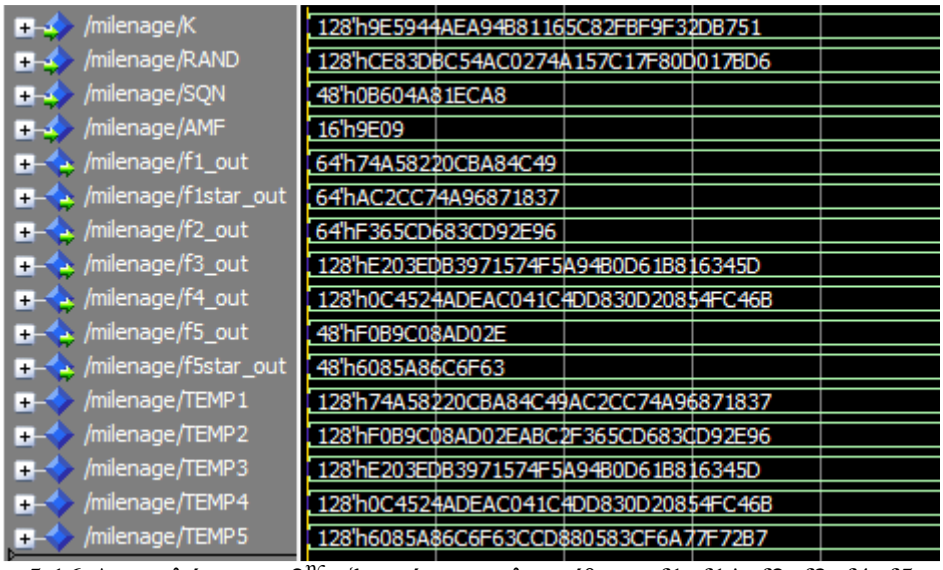
RAND: ce83dbc5 4ac0274a 157c17f8 0d017bd6

SQN : 0b604a81 eca8

AMF: 9e09

OP: 223014c5 806694c0 07ca1eee f57f004f

Τα αποτελέσματα της εξομοίωσης παρουσιάζονται στην παρακάτω εικόνα σε δεκαεξαδικό σύστημα. Τα τέσσερα πρώτα σήματα αναπαριστούν τις εισόδους των αλγορίθμων τις οποίες έχουμε ορίσει. Στην συνέχεια αναπαριστούνται τα σήματα TEMP, τα οποία αποτελούν ενδιάμεσα σήματα για την υλοποίηση των αλγορίθμων και τέλος εμφανίζονται τα αποτελέσματα του κάθε αλγορίθμου του MILENAGE σε δεκαεξαδικό σύστημα.



Εικόνα 5.16 Αποτελέσματα 2^{ης} εξομοίωσης αλγορίθμων f1, f1*, f2, f3, f4, f5 και f5*

f1_out: 74a58220 cba84c49

f1*_out: ac2cc74a 96871837

f2_out: f365cd68 3cd92e96

f5_out: f0b9c08a d02e

f3_out: e203edb3 971574f5 a94b0d61 b816345d

f4_out: 0c4524ad eac041c4 dd830d20 854fc46b

f5*_out: 6085a86c 6f6

ΚΕΦΑΛΑΙΟ 6
ΥΛΟΠΟΙΗΣΗ ΜΗΧΑΝΙΣΜΟΥ MILENAGE

6.1 Υλοποίηση με Χρήση FPGAs

Όπως αναφέρθηκε και σε προηγούμενο κεφάλαιο, η χρήση του FPGA παρέχει μεγαλύτερη ευελιξία και μπορούμε εύκολα και χωρίς μεγάλο κόστος να υλοποιήσουμε ένα ολοκληρωμένο κύκλωμα. Για αυτό το λόγο, η υλοποίηση του μηχανισμού ασφαλείας MILENAGE στην παρούσα πτυχιακή εργασία έχει γίνει με την χρήση της συστοιχίας επιτόπια προγραμματιζόμενων πυλών, FPGA.

Με τα FPGAs μπορούμε ουσιαστικά να 'μεταφράσουμε' τον εξομοιωμένο κώδικά μας σε πραγματικό υλικό, δηλαδή σε πύλες και δομές. Ο κώδικας με τον οποίο μπορούμε να προγραμματίσουμε ένα FPGA γράφεται σε γλώσσα περιγραφής υλικού, στην παρούσα πτυχιακή εργασία χρησιμοποιούμε VHDL.

6.2 Σύνθεση – Αποτελέσματα

Ο μηχανισμός ασφαλείας MILENAGE, ο οποίος υλοποιήθηκε με γλώσσα περιγραφής υλικού VHDL, εισήχθη στο πρόγραμμα σύνθεσης. Κατά την σύνθεση δημιουργείται μία λογική έκφραση για κάθε λογική συνάρτηση του κυκλώματος, απεικονίζεται ο σχεδιασμός σε μία προγραμματιζόμενη διάταξη (FPGA) και δημιουργούνται αρχεία για τις εξόδους της λειτουργίας προσομοίωσης και για τον προγραμματισμό των διατάξεων καθώς υλοποιείται χρονική ανάλυση.

Ο κώδικας εισήχθη στο εργαλείο σύνθεσης ISE Design Suite 14.7. Η σύνθεση υλοποιήθηκε με χρήση του FPGA Virtex-7 VC707 Evaluation Platform της οικογένειας Virtex7. Το Virtex αποτελεί την ναυαρχίδα της οικογένειας των προϊόντων FPGAs τα οποία έχουν αναπτυχθεί από την Xilinx.

Το Virtex®-7 FPGA VC707 Evaluation Kit είναι μία πλήρως εξοπλισμένη, με μεγάλη ευελιξία και υψηλής ταχύτητας πλατφόρμα η οποία χρησιμοποιεί το Virtex-7 XC7VX485T - 2FFG1761C και περιλαμβάνει βασικά συστατικά του hardware, εργαλεία σχεδιασμού, IP, σειριακή σύνδεση και προηγμένη διασύνδεση μνήμης.

Τα FPGA της σειράς Virtex 7 δομούνται από LUTs 6 εισόδων. Αυτά τα LUTs των 6 εισόδων μπορούν να χρησιμοποιηθούν είτε ως άπλα 6-LUTs, είτε ως δυο 5-LUTs εφόσον τα τελευταία μοιράζονται τις ίδιες εισόδους. Τα Virtex 7 συνδυάζουν 4 λογικά κελιά, μαζί με τα flip-flops στις εξόδους των κελιών, για να δημιουργήσουν ένα slice και διαθέτει συνολικά 75.900 slices.

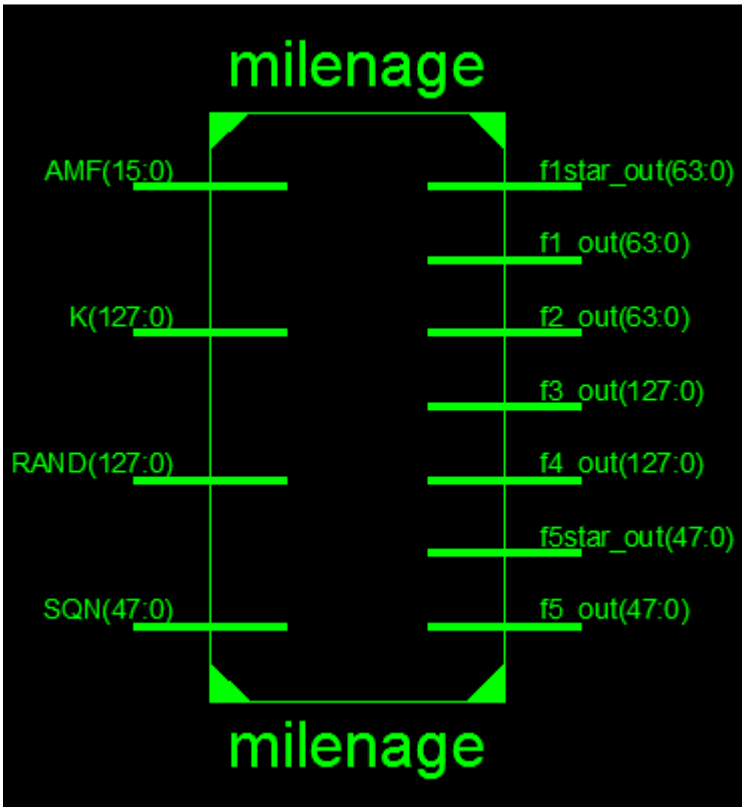
Στην συνέχεια αναφέρονται τα αποτελέσματα της σύνθεσης του συστήματος.

Evaluation Development Board	Virtex-7 VC707 Evaluation Platform
Product Category	All
Family	Virtex7
Device	XC7VX485T
Package	FFG1761
Speed	-2
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-PE VHDL
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-200X

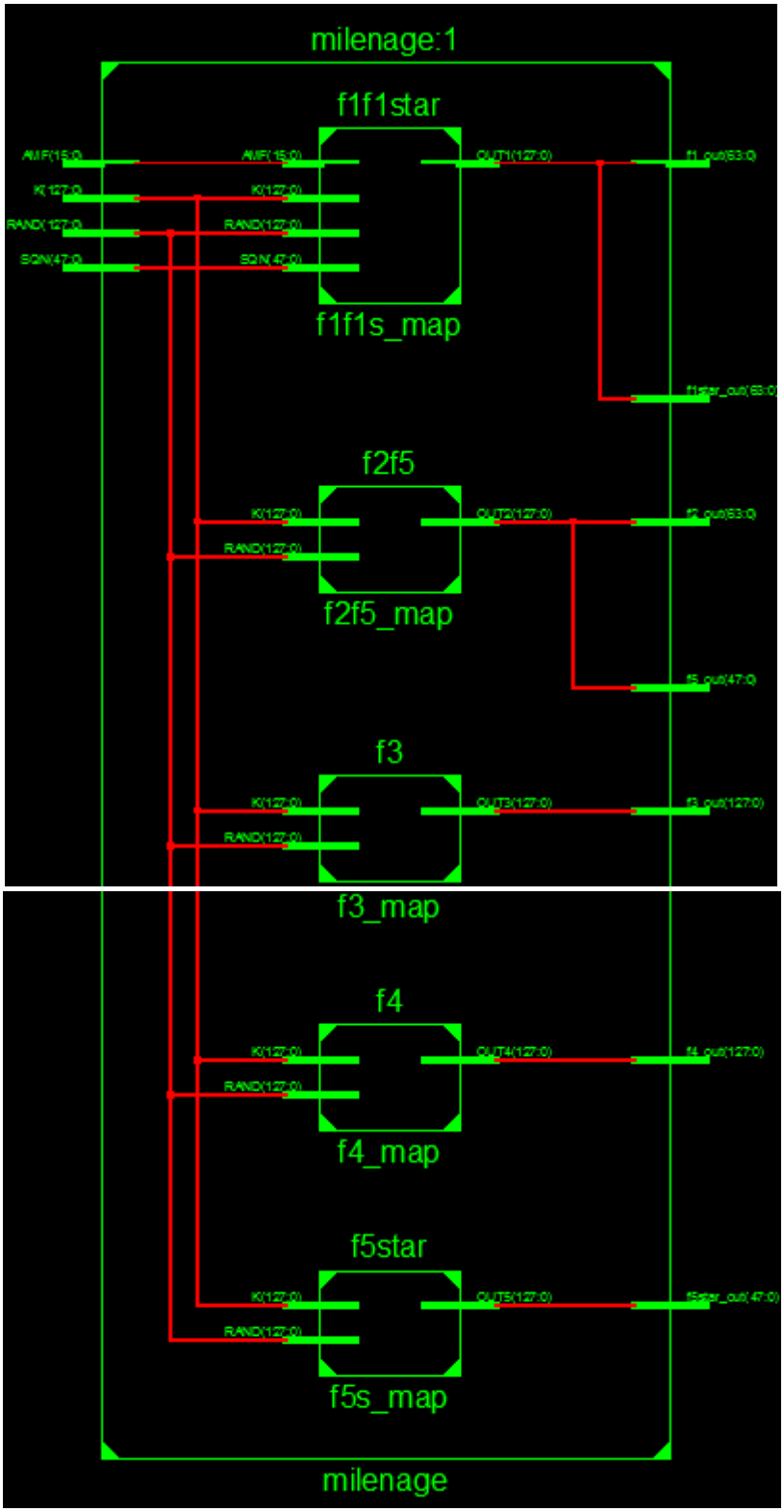
Εικόνα 6.1 Ιδιότητες Σύνθεσης MILENAGE

```
=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                                  : 3000
 256x8-bit single-port distributed Read Only RAM       : 3000
# Multiplexers                                          : 4320
 8-bit 2-to-1 multiplexer                              : 4320
# Xors                                                  : 13856
 128-bit xor2                                          : 26
 8-bit xor2                                            : 9510
 8-bit xor4                                            : 2160
 9-bit xor2                                            : 2160

=========================================================================
=========================================================================
*                            Design Summary
=========================================================================

Top Level Output File Name         : milenage.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                             : 187320
#      GND                         : 1
#      LUT2                        : 29388
#      LUT3                        : 736
#      LUT4                        : 576
#      LUT5                        : 11188
#      LUT6                        : 86198
#      MUXF7                       : 39488
#      MUXF8                       : 19744
#      VCC                         : 1
# IO Buffers                       : 864
#      IBUF                        : 320
#      OBUF                        : 544

Device utilization summary:
---------------------------

Selected Device : 7vx485tffg1761-2

Slice Logic Utilization:
 Number of Slice LUTs:                  128086  out of  303600    42%
    Number used as Logic:               128086  out of  303600    42%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    128086
   Number with an unused Flip Flop:     128086  out of  128086   100%
   Number with an unused LUT:                0  out of  128086     0%
   Number of fully used LUT-FF pairs:        0  out of  128086     0%
   Number of unique control sets:            0

IO Utilization:
 Number of IOs:                           864
 Number of bonded IOBs:                   864  out of    700   123% (*)

Specific Feature Utilization:
```

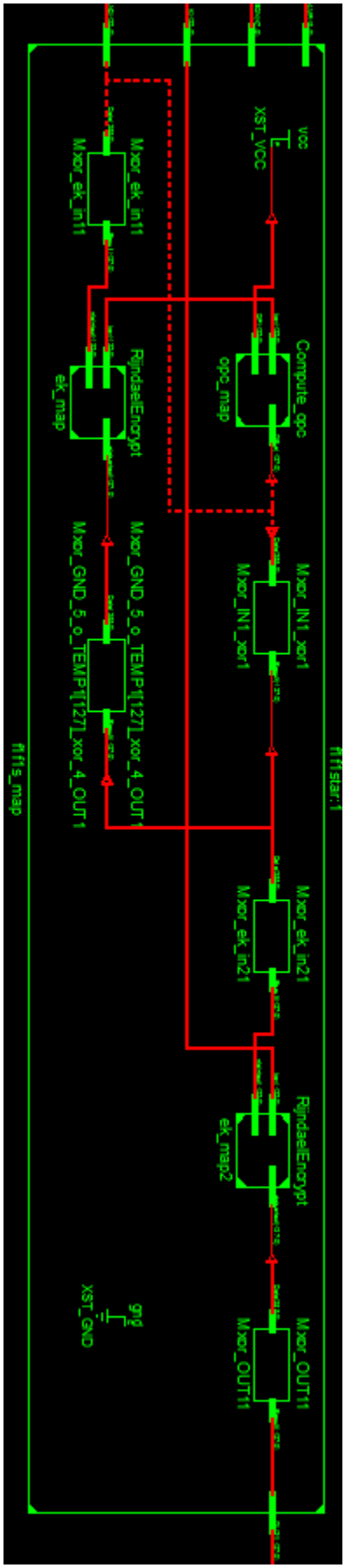
Εικόνα 6.2 Αποτελέσματα Σύνθεσης MILENAGE



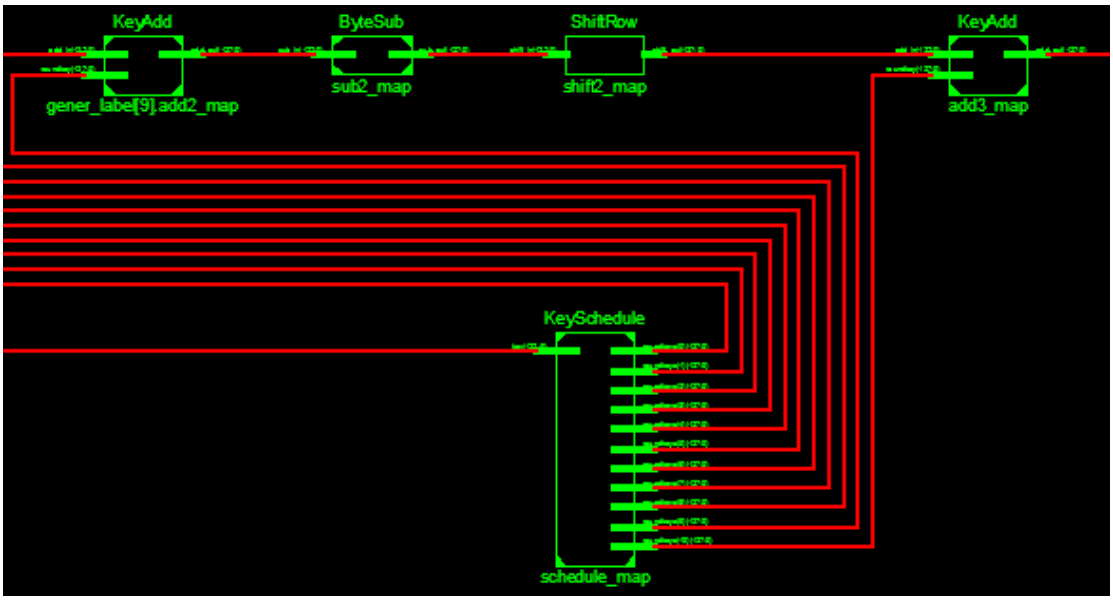
Εικόνα 6.3 Σύνθεση MILENAGE



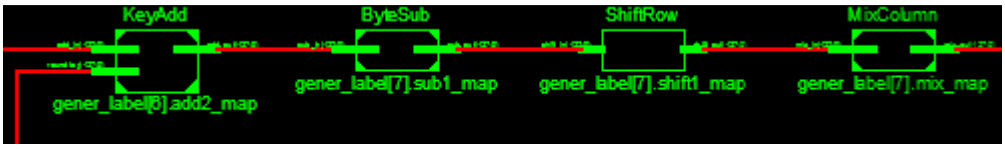
Εικόνα 6.4 Διάταξη Αλγορίθμων MILENAGE



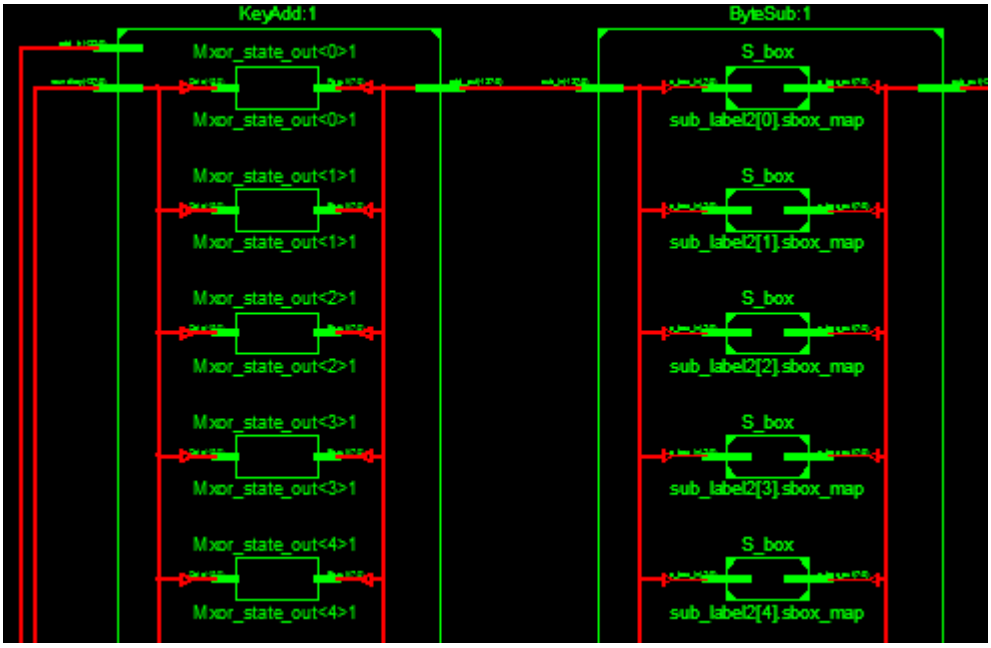
Εικόνα 6.5 Διάταξη f1, f5



Εικόνα 6.6 Μέρος Διάταξης Εκ Α



Εικόνα 6.7 Μέρος Διάταξης Εκ Β



Εικόνα 6.8 Μέρος Διάταξης Εκ Γ

ΕΠΙΛΟΓΟΣ

Σκοπός της πτυχιακής εργασίας ήταν η υλοποίηση του μηχανισμού ασφαλείας MILENAGE, με βάση τον αλγόριθμο Rijndael, για δίκτυα Τέταρτης Γενιάς (4G) με χρήση επιτόπια προγραμματιζόμενων πυλών, FPGA. Ανέδειξε ότι πρόκειται για ένα ολοκληρωμένο κύκλωμα, όπου το υλικό του είναι σε θέση να τροποποιηθεί κατάλληλα, ώστε να παρέχει την ασφάλεια η οποία είναι αναγκαία για την λειτουργία των ασύρματων δικτύων.

Αρχικά έγινε μια ιστορική αναφορά στα προγενέστερα δίκτυα όπου αναλύθηκαν τεχνικά χαρακτηρίστηκα και εφαρμογές αλλά και οι ελλείψεις οι οποίες συντέλεσαν στην ανάπτυξη των δικτύων Τέταρτης Γενιάς. Στην συνέχεια, αναφέρθηκαν οι στόχοι και οι προκλήσεις των δικτύων Τέταρτης Γενιάς, οι τεχνολογίες οι οποίες αναπτύχθηκαν και δόθηκε μεγάλη σημασία στην ασφάλεια η οποία παρέχεται. Πιο συγκεκριμένα, έγινε ανάλυση στην αρχιτεκτονική των 4G δικτύων με εκτενή αναφορά στις 8 διαστάσεις ασφαλείας, στις συναρτήσεις ασφαλείας αλλά και στους 7 αλγορίθμους – λειτουργίες, οι οποίοι αποτελούν την δομή του μηχανισμού ασφαλείας MILENAGE.

Οι αλγόριθμοι κρυπτογράφησης αποτέλεσαν ένα μεγάλο κεφάλαιο κατά την ανάπτυξη της πτυχιακής εργασίας, όπου αρχικά έγινε ανάλυση της έννοιας της κρυπτογραφίας, ορίστηκαν οι απαιτήσεις για την ανάπτυξη ενός κρυπτογραφικού αλγόριθμου και βασικές έννοιες όπως επίσης και τα είδη των αλγορίθμων κρυπτογραφίας και μηχανισμών κρυπτογράφησης. Αφού αναφέρθηκαν περιληπτικά βασικοί αλγόριθμοι κρυπτογράφησης έγινε αναλυτική αναφορά στον αλγόριθμο Rijndael, ψηφισμένος ως AES, αλλά και στην κάθε μία από τις λειτουργίες του. Σημαντική αναφορά αποτελεί η ασφάλεια του αλγορίθμου Rijndael.

Στο τρίτο κεφάλαιο, παρουσιάστηκαν και αναλύθηκαν οι μέθοδοι υλοποίησης ολοκληρωμένων κυκλωμάτων, FPGAs και ASICs. Για τα FPGAs έγινε περαιτέρω ανάλυση σχετικά με την δομή, τα χαρακτηριστικά αλλά και τον τρόπο λειτουργίας τους και της αρχιτεκτονικής. Παρουσιάστηκαν δηλαδή τα λογικά μπλοκ, τα κανάλια διασύνδεσης και οι ακίδες εισόδου – εξόδου.

Στην συνέχεια, υλοποιήθηκε ο σχεδιασμός του μηχανισμού ασφαλείας MILENAGE, όπου ορίσθηκαν αρχικά για κάθε αλγόριθμο, από τους οποίους αποτελείται, οι λειτουργίες του και οι αντίστοιχοι είσοδοι και έξοδοι. Ως βάση του μηχανισμού επιλέχθηκε ο αλγόριθμος κρυπτογράφησης Rijndael. Όπως επιβεβαιώθηκε, ο Rijndael αποτελεί την πιο ασφαλή επιλογή καθώς αποδεδειγμένα αποτελεί έναν από τους πιο ασφαλείς αλγορίθμους κρυπτογράφησης. Κάθε αλγόριθμος του MILENAGE αποτελείται από συστατικά και ο ορισμός τους καθώς και ο τρόπος διασύνδεσης τους παρουσιάστηκε αναλυτικά με τα πλαίσια του MILENAGE και του αλγορίθμου Rijndael.

Ο κώδικας του μηχανισμού ασφαλείας αναπτύχθηκε με την γλώσσα περιγραφής υλικού VHDL, όπου αρχικά έγινε αναφορά στην VHDL, τα χαρακτηριστικά της, τα πεδία εφαρμογής καθώς και στα δομικά στοιχεία από τα οποία αποτελείται. Στην συνέχεια περιγράφεται εκτενώς κάθε συστατικό των αλγορίθμων του μηχανισμού ασφαλείας. Παρατηρήθηκε ότι η VHDL οδήγησε με σχετική ευκολία στη σχεδίαση και την υλοποίηση του μηχανισμού ασφαλείας MILENAGE καθώς η δυνατότητα σχεδίασης με βάση τη συμπεριφορά και όχι την δομή παρέχει το πλεονέκτημα ότι ο σχεδιαστής μπορεί εύκολα να επικεντρωθεί στην συμπεριφορά την ο οποία ορίζει. Ο κώδικας παρουσιάζεται αναλυτικά στο Παράρτημα Ι.

Για την ανάλυση του κώδικα και τον συντακτικό έλεγχο έγινε η μετάφραση – compile με το εργαλείο ModelSim PE Student Edition 10.4.a. Αφού ολοκληρώθηκε επιτυχώς η διαδικασία της μετάφρασης στην συνέχεια έγινε ο έλεγχος ορθής λειτουργίας με την χρήση των dofiles. Σε κάθε είσοδο του MILENAGE δόθηκαν τιμές και αφού έγινε η εξομοίωση του κώδικα αναλύθηκαν τα αποτελέσματα δηλαδή οι έξοδοι. Επιβεβαιώθηκε βάση των αποτελεσμάτων η ορθότητα του κώδικα που έχει αναπτυχθεί με την χρήση δύο διαφορετικών σετ εισόδων για τους αλγόριθμους.

Η υλοποίηση του μηχανισμού ασφαλείας ολοκληρώθηκε με την διαδικασία της σύνθεσης του ολοκληρωμένου κυκλώματος με την χρήση FPGA. Το εργαλείο που χρησιμοποιήθηκε για την σύνθεση είναι το ISE Design Suite 14.7 και το ολοκληρωμένο κύκλωμα το οποίο επιλέχτηκε είναι το FPGA Virtex-7 VC707 Evaluation Platform της οικογενείας Virtex7. Τα αποτελέσματα της σύνθεσης του μηχανισμού καθώς και οι διατάξεις των υλικών – πυλών παρουσιάζονται μαζί με τις αντίστοιχες διασυνδέσεις και τις εισόδους – εξόδους. Συμπεράνουμε ότι ο ιεραρχικός σχεδιασμός που παρέχει η VHDL είναι κατάλληλος για την επαναχρησιμοποίηση των υποκυκλωμάτων.

Με βάση τα αποτελέσματα της σύνθεσης ο μηχανισμός ασφαλείας MILENAGE αποτελεί ένα ευρύ ολοκληρωμένο κύκλωμα, το οποίο κάνει χρήση μεγάλου αριθμού υλικού όπως πύλες και καλώδια. Ο ιεραρχικός σχεδιασμός που ορίσαμε ¨χωρίζει¨ το κύκλωμα στα πολλά επίπεδα καθιστώντας ευκολότερο τον σχεδιασμό του. Αποτελείται από 128086 slices και από 864 ακίδες εισόδου – εξόδου.

ΠΑΡΑΡΤΗΜΑ 1
ΚΩΔΙΚΑΣ VHDL

Κώδικας MILENAGE

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
---------------------------------------

ENTITY milenage IS
PORT (K         : IN  std_logic_vector(127 downto 0);
     RAND      : IN  std_logic_vector(127 downto 0);
     SQN       : IN  std_logic_vector(47 downto 0);
     AMF       : IN  std_logic_vector(15 downto 0);
     f1_out    : OUT std_logic_vector(63 downto 0);
     f1star_out: OUT std_logic_vector(63 downto 0);
     f2_out    : OUT std_logic_vector(63 downto 0);
     f3_out    : OUT std_logic_vector(127 downto 0);
     f4_out    : OUT std_logic_vector(127 downto 0);
     f5_out    : OUT std_logic_vector(47 downto 0);
     f5star_out: OUT std_logic_vector(47 downto 0));
END milenage;

ARCHITECTURE structural OF milenage IS

COMPONENT f1f1star IS
PORT (K    : IN  std_logic_vector(127 downto 0);
     RAND : IN  std_logic_vector(127 downto 0);
     SQN  : IN  std_logic_vector(47 downto 0);
     AMF  : IN  std_logic_vector(15 downto 0);
     OUT1 : OUT std_logic_vector(127 downto 0));
END COMPONENT;

COMPONENT f2f5 IS
PORT (K    : IN  std_logic_vector(127 downto 0);
     RAND : IN  std_logic_vector(127 downto 0);
     OUT2 : OUT std_logic_vector(127 downto 0));
END COMPONENT;

COMPONENT f3 IS
PORT (K    : IN  std_logic_vector(127 downto 0);
     RAND : IN  std_logic_vector(127 downto 0);
     OUT3 : OUT std_logic_vector(127 downto 0));
```

```
END COMPONENT;

COMPONENT f4 IS
PORT (K    : IN  std_logic_vector(127 downto 0);
     RAND : IN  std_logic_vector(127 downto 0);
     OUT4 : OUT std_logic_vector(127 downto 0));
END COMPONENT;

COMPONENT f5star IS
PORT (K    : IN  std_logic_vector(127 downto 0);
     RAND : IN  std_logic_vector(127 downto 0);
     OUT5 : OUT std_logic_vector(127 downto 0));
END COMPONENT;

SIGNAL TEMP1, TEMP2, TEMP3, TEMP4, TEMP5 : std_logic_vector(127 downto
0);

BEGIN

f1f1s_map : f1f1star PORT MAP (K    => K,
                        RAND => RAND,
                        SQN  => SQN,
                        AMF  => AMF,
                        OUT1 => TEMP1);

f2f5_map : f2f5 PORT MAP (K    => K,
                     RAND => RAND,
                     OUT2 => TEMP2);

f3_map : f3 PORT MAP (K    => K,
                  RAND => RAND,
                  OUT3 => TEMP3);

f4_map : f4 PORT MAP (K    => K,
                  RAND => RAND,
                  OUT4 => TEMP4);

f5s_map : f5star PORT MAP (K    => K,
                      RAND => RAND,
                      OUT5 => TEMP5);

f1_out <= TEMP1(127 downto 64);
f1star_out <= TEMP1(63 downto 0);
f2_out <= TEMP2(63 downto 0);
```

```
f3_out <= TEMP3(127 downto 0);
f4_out <= TEMP4(127 downto 0);
f5_out <= TEMP2(127 downto 80);
f5star_out <= TEMP5(127 downto 80);

END structural;
```

Κώδικας f1 και f1*

```
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-------------------------------------

ENTITY f1f1star IS
PORT (K    : IN  std_logic_vector(127 downto 0);
     RAND : IN  std_logic_vector(127 downto 0);
     SQN  : IN  std_logic_vector(47 downto 0);
     AMF  : IN  std_logic_vector(15 downto 0);
     OUT1 : OUT std_logic_vector(127 downto 0));
END f1f1star;

ARCHITECTURE structural OF f1f1star IS

COMPONENT Compute_opc IS
PORT (OP   : IN  std_logic_vector(127 downto 0);
     key  : IN  std_logic_vector(127 downto 0);
     OP_c : OUT std_logic_vector(127 downto 0));
END COMPONENT;

COMPONENT RijndaelEncrypt IS
PORT (plaintext : IN  std_logic_vector(127 downto 0);
     key       : IN  std_logic_vector(127 downto 0);
     ciphertext: OUT std_logic_vector(127 downto 0));
END COMPONENT;

SIGNAL OP_c, ek_in1, IN1, IN1_xor, TEMP1, rot_out, ek_in2, TEMP2 :
std_logic_vector(127 downto 0);
CONSTANT c1 : std_logic_vector(127 downto 0) :=
"00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000";
CONSTANT OP : std_logic_vector(127 downto 0) :=
"11001101110000100000001011010101000100100011111000100000111101100010
10110110110101100111011010101100011100101100101100110001 1000";
```

```
BEGIN

  opc_map : Compute_opc PORT MAP (OP  => OP,
                                  key  => K,
                                  OP_c => OP_c);

  ek_in1 <= OP_c XOR RAND;

  ek_map : RijndaelEncrypt PORT MAP (plaintext  => ek_in1,
                                     key        => K,
                                     ciphertext => TEMP1);

  IN1 <= SQN & AMF & SQN & AMF;
  IN1_xor <=  IN1 XOR OP_c;

  --rotate by r1 : 64
  rot_out <= IN1_xor(63 DOWNTO 0) & IN1_xor (127 DOWNTO 64);

  ek_in2 <= c1 XOR TEMP1 XOR rot_out;

  ek_map2 : RijndaelEncrypt PORT MAP (plaintext  => ek_in2,
                                      key        => K,
                                      ciphertext => TEMP2);

  OUT1 <= OP_c XOR TEMP2;

END structural;
```

Κώδικας f2 και f5

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
---------------------------------------

ENTITY f2f5 IS
PORT (K    : IN  std_logic_vector(127 downto 0);
      RAND : IN  std_logic_vector(127 downto 0);
      OUT2 : OUT std_logic_vector(127 downto 0));
END f2f5;

ARCHITECTURE structural OF f2f5 IS

COMPONENT Compute_opc IS
```

```
PORT (OP   : IN  std_logic_vector(127 downto 0);
     key  : IN  std_logic_vector(127 downto 0);
     OP_c : OUT std_logic_vector(127 downto 0));
END COMPONENT;

COMPONENT RijndaelEncrypt IS
PORT (plaintext : IN  std_logic_vector(127 downto 0);
     key       : IN  std_logic_vector(127 downto 0);
     ciphertext: OUT std_logic_vector(127 downto 0));
END COMPONENT;

SIGNAL OP_c, ek_in1, TEMP1, rot_in, rot_out, ek_in2, TEMP2 :
std_logic_vector(127 downto 0);
CONSTANT c2 : std_logic_vector(127 downto 0) :=
"00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000001";
CONSTANT OP : std_logic_vector(127 downto 0) :=
"11001101110000100000001011010101000100100011111000100000111101100010
10110110110101100111011010101100011100101100101100110001 1000";

BEGIN

  opc_map : Compute_opc PORT MAP (OP  => OP,
                            key  => K,
                            OP_c => OP_c);

  ek_in1 <= OP_c XOR RAND;

  ek_map : RijndaelEncrypt PORT MAP (plaintext  => ek_in1,
                               key        => K,
                               ciphertext => TEMP1);

  rot_in <= TEMP1 XOR OP_c;

  --rotate by r2 : 0
  rot_out <= rot_in;

  ek_in2 <= c2 XOR rot_out;

  ek_map2 : RijndaelEncrypt PORT MAP (plaintext  => ek_in2,
                               key        => K,
                                ciphertext => TEMP2);

  OUT2 <= OP_c XOR TEMP2;
```

```
END structural;
```

Κώδικας f3

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-------------------------------------

ENTITY f3 IS
PORT (K    : IN  std_logic_vector(127 downto 0);
     RAND : IN  std_logic_vector(127 downto 0);
     OUT3 : OUT std_logic_vector(127 downto 0));
END f3;

ARCHITECTURE structural OF f3 IS

COMPONENT Compute_opc IS
PORT (OP   : IN  std_logic_vector(127 downto 0);
     key  : IN  std_logic_vector(127 downto 0);
     OP_c : OUT std_logic_vector(127 downto 0));
END COMPONENT;

COMPONENT RijndaelEncrypt IS
PORT (plaintext : IN  std_logic_vector(127 downto 0);
     key       : IN  std_logic_vector(127 downto 0);
     ciphertext: OUT std_logic_vector(127 downto 0));
END COMPONENT;

SIGNAL OP_c, ek_in1, TEMP1, rot_out, ek_in2, TEMP2, rot_in :
std_logic_vector(127 downto 0);
CONSTANT c3 : std_logic_vector(127 downto 0) :=
"00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000010";
CONSTANT OP : std_logic_vector(127 downto 0) :=
"11001101110000100000001011010101000100100011111000100000111101100010
10110110110101100111011010101100011100101100101100110001 1000";

BEGIN

  opc_map : Compute_opc PORT MAP (OP  => OP,
                                  key  => K,
                                  OP_c => OP_c);
```

```
  ek_in1 <= OP_c XOR RAND;

  ek_map : RijndaelEncrypt PORT MAP (plaintext  => ek_in1,
                                key        => K,
                                ciphertext => TEMP1);

  rot_in <= TEMP1 XOR OP_c;

  --rotate by r3 : 32
  rot_out <= rot_in(95 DOWNTO 0) & rot_in(127 DOWNTO 96);

  ek_in2 <= c3 XOR rot_out;

  ek_map2 : RijndaelEncrypt PORT MAP (plaintext  => ek_in2,
                                key        => K,
                                ciphertext => TEMP2);

  OUT3 <= OP_c XOR TEMP2;

END structural;
```

Κώδικας f4

```
USE ieee.std_logic_1164.all;
USE work.my_package.all;
---------------------------------------

ENTITY f4 IS
PORT (K    : IN  std_logic_vector(127 downto 0);
     RAND : IN  std_logic_vector(127 downto 0);
     OUT4 : OUT std_logic_vector(127 downto 0));
END f4;

ARCHITECTURE structural OF f4 IS

COMPONENT Compute_opc IS
PORT (OP   : IN  std_logic_vector(127 downto 0);
     key  : IN  std_logic_vector(127 downto 0);
     OP_c : OUT std_logic_vector(127 downto 0));
END COMPONENT;

COMPONENT RijndaelEncrypt IS
PORT (plaintext : IN  std_logic_vector(127 downto 0);
```

```
    key       : IN  std_logic_vector(127 downto 0);
    ciphertext: OUT std_logic_vector(127 downto 0));
END COMPONENT;


SIGNAL OP_c, ek_in1, TEMP1, rot_out, ek_in2, TEMP2, rot_in :
std_logic_vector(127 downto 0);
CONSTANT c4 : std_logic_vector(127 downto 0) :=
"00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000100";
CONSTANT OP : std_logic_vector(127 downto 0) :=
"11001101110000100000001011010101000100100011111000100000111101100010
10110110110101100111011010101100011100101100101100110001 1000";

BEGIN

 opc_map : Compute_opc PORT MAP (OP  => OP,
                          key  => K,
                          OP_c => OP_c);

 ek_in1 <= OP_c XOR RAND;

 ek_map : RijndaelEncrypt PORT MAP (plaintext  => ek_in1,
                            key        => K,
                            ciphertext => TEMP1);

 rot_in <= TEMP1 XOR OP_c;

 --rotate by r4 : 64
 rot_out <= rot_in(63 DOWNTO 0) & rot_in(127 DOWNTO 64);

 ek_in2 <= c4 XOR rot_out;

 ek_map2 : RijndaelEncrypt PORT MAP (plaintext  => ek_in2,
                           key        => K,
                           ciphertext => TEMP2);

 OUT4 <= OP_c XOR TEMP2;

END structural;
```

Κώδικας f5star

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-------------------------------------

ENTITY f5star IS
PORT (K    : IN  std_logic_vector(127 downto 0);
     RAND : IN  std_logic_vector(127 downto 0);
     OUT5 : OUT std_logic_vector(127 downto 0));
END f5star;

ARCHITECTURE structural OF f5star IS

COMPONENT Compute_opc IS
PORT (OP   : IN  std_logic_vector(127 downto 0);
     key  : IN  std_logic_vector(127 downto 0);
     OP_c : OUT std_logic_vector(127 downto 0));
END COMPONENT;

COMPONENT RijndaelEncrypt IS
PORT (plaintext : IN  std_logic_vector(127 downto 0);
     key       : IN  std_logic_vector(127 downto 0);
     ciphertext: OUT std_logic_vector(127 downto 0));
END COMPONENT;

SIGNAL OP_c, ek_in1, TEMP1, rot_out, ek_in2, TEMP2, rot_in : std_logic_vector(127
downto 0);
CONSTANT c5 : std_logic_vector(127 downto 0) :=
"00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000001000";
CONSTANT OP : std_logic_vector(127 downto 0) :=
"11001101110000100000001011010101000100100011111000100000111101100010101101
10110101100111011010101100011100101100101100110000 1000";
CONSTANT r5 : INTEGER := 96;

BEGIN

  opc_map : Compute_opc PORT MAP (OP  => OP,
                            key  => K,
                            OP_c => OP_c);

  ek_in1 <= OP_c XOR RAND;

  ek_map : RijndaelEncrypt PORT MAP (plaintext  => ek_in1,
                               key        => K,
                               ciphertext => TEMP1);
```

```
  rot_in <= TEMP1 XOR OP_c;


  --rotate by r5 : 96
  rot_out <= rot_in(31 DOWNTO 0) & rot_in(127 DOWNTO 32);

  ek_in2 <= c5 XOR rot_out;

  ek_map2 : RijndaelEncrypt PORT MAP (plaintext  => ek_in2,
                                      key        => K,
                                      ciphertext => TEMP2);

  OUT5 <= OP_c XOR TEMP2;

END structural;
```

Κώδικας RijndaelEncrypt

```
USE ieee.std_logic_1164.all;
USE work.my_package.all;
---------------------------------------

ENTITY RijndaelEncrypt IS
PORT (plaintext  : IN  std_logic_vector(127 downto 0);
      key    : IN std_logic_vector(127 downto 0);
      ciphertext : OUT  std_logic_vector(127 downto 0));
END RijndaelEncrypt;
-----------------------------------------------

ARCHITECTURE structural OF RijndaelEncrypt IS

COMPONENT KeySchedule
PORT (key       : IN  std_logic_vector(127 downto 0);
      roundkeys : OUT key_matrix);
END COMPONENT;

COMPONENT KeyAdd
PORT (add_in   : IN  std_logic_vector(127 downto 0);
      roundkey : IN  std_logic_vector(127 downto 0);
      add_out  : OUT  std_logic_vector(127 downto 0));
END COMPONENT;

COMPONENT ByteSub
PORT (sub_in : IN std_logic_vector(127 DOWNTO 0);
      sub_out: OUT std_logic_vector(127 DOWNTO 0));
END COMPONENT;
```

```
COMPONENT ShiftRow
PORT (shift_in : IN std_logic_vector(127 DOWNTO 0);
     shift_out: OUT std_logic_vector(127 DOWNTO 0));
END COMPONENT;

COMPONENT MixColumn
PORT (mix_in : IN std_logic_vector(127 DOWNTO 0);
     mix_out: OUT std_logic_vector(127 DOWNTO 0));
END COMPONENT;

SIGNAL round_keys: key_matrix;
SIGNAL sub_out, shift_out, add_out: key_matrix2;
SIGNAL mix_out: key_matrix3;


BEGIN

  schedule_map: KeySchedule PORT MAP (key  => key,
                                roundkeys => round_keys);

  add1_map: KeyAdd PORT MAP (add_in   => plaintext,
                        roundkey => round_keys(0),
                        add_out  => add_out(1));

  gener_label: FOR i IN 1 TO 9 GENERATE

    sub1_map  : ByteSub   PORT MAP (sub_in  => add_out(i),
                              sub_out => sub_out(i));
    shift1_map: ShiftRow  PORT MAP (shift_in  => sub_out(i),
                              shift_out => shift_out(i));
    mix_map   : MixColumn PORT MAP (mix_in  => shift_out(i),
                              mix_out => mix_out(i));
    add2_map  : KeyAdd    PORT MAP (add_in   => mix_out(i),
                              roundkey => round_keys(i),
                              add_out  => add_out(i+1));
  END GENERATE gener_label;

  sub2_map  : ByteSub   PORT MAP (sub_in  => add_out(10),
                            sub_out => sub_out(10));
  shift2_map: ShiftRow  PORT MAP (shift_in  => sub_out(10),
                            shift_out => shift_out(10));
  add3_map  : KeyAdd    PORT MAP (add_in   => shift_out(10),
                            roundkey => round_keys(10),
```

```
                    add_out  => ciphertext);


END structural;
```

Κώδικας KeyAdd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-------------------------------------

ENTITY KeyAdd IS
PORT (add_in   : IN  std_logic_vector(127 downto 0);
     roundkey : IN std_logic_vector(127 downto 0);
     add_out  : OUT  std_logic_vector(127 downto 0));
END KeyAdd;

ARCHITECTURE behavioral OF KeyAdd IS

SIGNAL state, state_out, roundkey_array : row;

BEGIN

   add_lab:
   FOR i IN 15 DOWNTO 0 GENERATE
              state(15-i) <= add_in(8*i+7 DOWNTO 8*i);
              roundkey_array(15-i) <= roundkey(8*i+7 DOWNTO 8*i);
             state_out(i) <= state(i) XOR roundkey_array(i);
              add_out(8*i+7 DOWNTO 8*i) <= state_out(15-i);
   END GENERATE add_lab;

END behavioral;
```

Κώδικας ByteSub

```
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-----------------------------------

ENTITY ByteSub IS
PORT (sub_in : IN std_logic_vector(127 DOWNTO 0);
     sub_out: OUT std_logic_vector(127 DOWNTO 0));
END ByteSub;
-----------------------------------

ARCHITECTURE structural OF ByteSub IS

COMPONENT S_box IS
  PORT (s_box_in : IN std_logic_vector(7 DOWNTO 0);
       s_box_out: OUT std_logic_vector(7 DOWNTO 0));
END COMPONENT;

SIGNAL state : row;

BEGIN

 sub_label:FOR i IN 15 DOWNTO 0 GENERATE
                state(15-i) <= sub_in(8*i+7 DOWNTO 8*i);
                END GENERATE sub_label;

sub_label2:FOR i IN 15 DOWNTO 0 GENERATE
         sbox_map: entity work.S_box PORT MAP (s_box_in  => state(15-i),
                          s_box_out => sub_out(8*i+7 DOWNTO 8*i));
                END GENERATE sub_label2;

END structural;
```

Component ShiftRows

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-----------------------------------

ENTITY ShiftRow IS
PORT (shift_in : IN std_logic_vector(127 DOWNTO 0);
     shift_out: OUT std_logic_vector(127 DOWNTO 0));
```

```
END ShiftRow;
-----------------------------------

ARCHITECTURE behavioral OF ShiftRow IS

SIGNAL state, state_out : row;

BEGIN

   shift_lab:
   FOR i IN 15 DOWNTO 0 GENERATE
         state(15-i) <= shift_in(8*i+7 DOWNTO 8*i);
   END GENERATE shift_lab;

   -- row 0 rotate by 0
   state_out(0)  <= state(0);
   state_out(4)  <= state(4);
   state_out(8)  <= state(8);
   state_out(12) <= state(12);
   -- row 1 rotate by 1
   state_out(1)  <= state(5);
   state_out(5)  <= state(9);
   state_out(9)  <= state(13);
   state_out(13) <= state(1);
   -- row 2 rotate by 2
   state_out(2)  <= state(10);
   state_out(10) <= state(2);
   state_out(6)  <= state(14);
   state_out(14) <= state(6);
   -- row 3 rotate by 3
   state_out(3)  <= state(15);
   state_out(15) <= state(11);
   state_out(11) <= state(7);
   state_out(7)  <= state(3);

   shift_lab1:
   FOR i IN 15 DOWNTO 0 GENERATE
         shift_out(8*i+7 DOWNTO 8*i) <= state_out(15-i);
   END GENERATE shift_lab1;

END behavioral;
```

Component MixColumns

```
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-----------------------------------

ENTITY MixColumn IS
PORT (mix_in : IN std_logic_vector(127 DOWNTO 0);
     mix_out: OUT std_logic_vector(127 DOWNTO 0));
END MixColumn;
-----------------------------------

ARCHITECTURE behavioral OF MixColumn IS

SIGNAL state, state_out, mult_2, mult_3: row;
SIGNAL shift_2, shift_3, xored: mix_array;

BEGIN

  mix_lab1:
  FOR i IN 15 DOWNTO 0 GENERATE
         state(15-i) <= mix_in(8*i+7 DOWNTO 8*i);
  END GENERATE mix_lab1;

  -- multiply by 2
  mix_lab2:
  FOR i IN  15 DOWNTO 0 GENERATE
   shift_2(i) <= state(i) & '0';
   WITH shift_2(i)(8) SELECT
     mult_2(i) <= shift_2(i)(7 DOWNTO 0) XOR "00011011" WHEN '1',
                  shift_2(i)(7 DOWNTO 0) WHEN '0',
                  "ZZZZZZZZ" WHEN OTHERS;
  END GENERATE mix_lab2;

  --multiply by 3
  mix_lab3:
  FOR i IN 15 DOWNTO 0 GENERATE
           shift_3(i) <= state(i) & '0';
           xored(i) <= shift_3(i) XOR '0' & state(i);
    WITH xored(i)(8) SELECT
     mult_3(i) <= xored(i)(7 DOWNTO 0) XOR "00011011" WHEN '1',
                  xored(i)(7 DOWNTO 0) WHEN '0',
                  "ZZZZZZZZ" WHEN OTHERS;
  END GENERATE mix_lab3;
```

```
--row 1
state_out(0)  <= mult_2(0)  XOR mult_3(1)  XOR state(2)  XOR state(3);
state_out(4)  <= mult_2(4)  XOR mult_3(5)  XOR state(6)  XOR state(7);
state_out(8)  <= mult_2(8)  XOR mult_3(9)  XOR state(10) XOR state(11);
state_out(12) <= mult_2(12) XOR mult_3(13) XOR state(14) XOR state(15);
--row 2
state_out(1)  <= state(0)  XOR mult_2(1)  XOR mult_3(2)  XOR state(3);
state_out(5)  <= state(4)  XOR mult_2(5)  XOR mult_3(6)  XOR state(7);
state_out(9)  <= state(8)  XOR mult_2(9)  XOR mult_3(10) XOR state(11);
state_out(13) <= state(12) XOR mult_2(13) XOR mult_3(14) XOR state(15);
--row 3
state_out(2)  <= state(0)  XOR state(1)  XOR mult_2(2)  XOR mult_3(3);
state_out(6)  <= state(4)  XOR state(5)  XOR mult_2(6)  XOR mult_3(7);
state_out(10) <= state(8)  XOR state(9)  XOR mult_2(10) XOR mult_3(11);
state_out(14) <= state(12) XOR state(13) XOR mult_2(14) XOR mult_3(15);
--row 4
state_out(3)  <= mult_3(0)  XOR state(1)  XOR state(2)  XOR mult_2(3);
state_out(7)  <= mult_3(4)  XOR state(5)  XOR state(6)  XOR mult_2(7);
state_out(11) <= mult_3(8)  XOR state(9)  XOR state(10) XOR mult_2(11);
state_out(15) <= mult_3(12) XOR state(13) XOR state(14) XOR mult_2(15);

 mix_lab4:
 FOR i IN 15 DOWNTO 0 GENERATE
         mix_out(8*i+7 DOWNTO 8*i) <= state_out(15-i);
 END GENERATE mix_lab4;

END behavioral;
```

Κώδικας KeySchedule

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-----------------------------------------

ENTITY KeySchedule IS
PORT (key       : IN  std_logic_vector(127 downto 0);
    roundkeys : OUT key_matrix);
END KeySchedule;
-----------------------------------------

ARCHITECTURE structural OF KeySchedule IS
```

```
COMPONENT S_box IS
PORT (s_box_in : IN std_logic_vector(7 DOWNTO 0);
     s_box_out: OUT std_logic_vector(7 DOWNTO 0));
END COMPONENT;

SIGNAL s_box_out1, s_box_out2, s_box_out3, s_box_out4 : round_array;
SIGNAL tmp_roundkey: matrix;
CONSTANT roundConst: round_array := ("00000001", "00000010", "00000100",
                              "00001000", "00010000", "00100000",
                              "01000000", "10000000", "00011011","00110110");
BEGIN

   tmp: FOR i IN 0 TO 15 GENERATE
            tmp_roundkey(0)(i) <= key(8*i+7 DOWNTO 8*i);
           END GENERATE tmp;

   temp: FOR i IN 1 TO 10 GENERATE
     sub_map1: S_box PORT MAP (s_box_in  => tmp_roundkey(i-1)(2),
                          s_box_out => s_box_out1(i));
     sub_map2: S_box PORT MAP (s_box_in  => tmp_roundkey(i-1)(1),
                          s_box_out => s_box_out2(i));
     sub_map3: S_box PORT MAP (s_box_in  => tmp_roundkey(i-1)(0),
                          s_box_out => s_box_out3(i));
     sub_map4: S_box PORT MAP (s_box_in  => tmp_roundkey(i-1)(3),
                          s_box_out => s_box_out4(i));
    END GENERATE temp;

  temp1: FOR i IN 1 TO 10 GENERATE
  tmp_roundkey(i)(15) <= s_box_out1(i) XOR tmp_roundkey(i-1)(15) XOR
roundConst(11-i);
  tmp_roundkey(i)(14) <= s_box_out2(i) XOR tmp_roundkey(i-1)(14);
  tmp_roundkey(i)(13) <= s_box_out3(i) XOR tmp_roundkey(i-1)(13);
  tmp_roundkey(i)(12) <= s_box_out4(i) XOR tmp_roundkey(i-1)(12);
  tmp_roundkey(i)(11) <= tmp_roundkey(i-1)(11) XOR tmp_roundkey(i)(15);
  tmp_roundkey(i)(10) <= tmp_roundkey(i-1)(10) XOR tmp_roundkey(i)(14);
  tmp_roundkey(i)(9)  <= tmp_roundkey(i-1)(9)  XOR tmp_roundkey(i)(13);
  tmp_roundkey(i)(8)  <= tmp_roundkey(i-1)(8)  XOR tmp_roundkey(i)(12);
  tmp_roundkey(i)(7)  <= tmp_roundkey(i-1)(7)  XOR tmp_roundkey(i)(11);
  tmp_roundkey(i)(6)  <= tmp_roundkey(i-1)(6)  XOR tmp_roundkey(i)(10);
  tmp_roundkey(i)(5)  <= tmp_roundkey(i-1)(5)  XOR tmp_roundkey(i)(9);
  tmp_roundkey(i)(4)  <= tmp_roundkey(i-1)(4)  XOR tmp_roundkey(i)(8);
  tmp_roundkey(i)(3)  <= tmp_roundkey(i-1)(3)  XOR tmp_roundkey(i)(7);
  tmp_roundkey(i)(2)  <= tmp_roundkey(i-1)(2)  XOR tmp_roundkey(i)(6);
  tmp_roundkey(i)(1)  <= tmp_roundkey(i-1)(1)  XOR tmp_roundkey(i)(5);
```

```
  tmp_roundkey(i)(0)  <= tmp_roundkey(i-1)(0)  XOR tmp_roundkey(i)(4);
  END GENERATE temp1;

    tmp1:
    FOR i IN 15 DOWNTO 0 GENERATE
          roundkeys(0)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(0)(i);
          roundkeys(1)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(1)(i);
          roundkeys(2)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(2)(i);
          roundkeys(3)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(3)(i);
          roundkeys(4)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(4)(i);
          roundkeys(5)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(5)(i);
          roundkeys(6)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(6)(i);
          roundkeys(7)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(7)(i);
          roundkeys(8)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(8)(i);
          roundkeys(9)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(9)(i);
          roundkeys(10)(8*i+7 DOWNTO 8*i) <= tmp_roundkey(10)(i);
        END GENERATE tmp1;

END structural;
```

Component Compute_opc

```
USE ieee.std_logic_1164.all;
USE work.my_package.all;
---------------------------------------

ENTITY Compute_opc IS
PORT (OP  : IN  std_logic_vector(127 downto 0);
    key : IN  std_logic_vector(127 downto 0);
    OP_c: OUT std_logic_vector(127 downto 0));
END Compute_opc;
---------------------------------------------

ARCHITECTURE structural OF Compute_opc IS

COMPONENT RijndaelEncrypt IS
PORT (plaintext  : IN  std_logic_vector(127 downto 0);
    key        : IN  std_logic_vector(127 downto 0);
    ciphertext : OUT std_logic_vector(127 downto 0));
END COMPONENT;

SIGNAL temp : std_logic_vector(127 downto 0);

BEGIN
```

```
   ek_map : RijndaelEncrypt PORT MAP (plaintext  => OP,
                                      key        => key,
                                      ciphertext => temp);

   OP_c <= temp XOR OP;

END structural;
```

Component Sbox

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
---------------------------------

ENTITY S_box IS
  PORT (s_box_in : IN std_logic_vector(7 DOWNTO 0);
       s_box_out: OUT std_logic_vector(7 DOWNTO 0));
END S_box;
-----------------------------------

ARCHITECTURE behavioral OF S_box IS

    BEGIN
    PROCESS(s_box_in)
     BEGIN
      CASE s_box_in IS
      --first row
      WHEN "00000000" => s_box_out <= "01100011"; --(X"63")
                    WHEN "00000001" => s_box_out <= "01111100"; --(X"7C")
                    WHEN "00000010" => s_box_out <= "01110111"; --(X"77")
                    WHEN "00000011" => s_box_out <= "01111011"; --(X"7B")
                    WHEN "00000100" => s_box_out <= "11110010"; --(X"F2")
                    WHEN "00000101" => s_box_out <= "01101011"; --(X"6B")
                    WHEN "00000110" => s_box_out <= "01101111"; --(X"6F")
                    WHEN "00000111" => s_box_out <= "11000101"; --(X"C5")
                    WHEN "00001000" => s_box_out <= "00110000"; --(X"30")
                    WHEN "00001001" => s_box_out <= "00000001"; --(X"01")
                    WHEN "00001010" => s_box_out <= "01100111"; --(X"67")
                    WHEN "00001011" => s_box_out <= "00101011"; --(X"2B")
                    WHEN "00001100" => s_box_out <= "11111110"; --(X"FE")
                    WHEN "00001101" => s_box_out <= "11010111"; --(X"D7")
                    WHEN "00001110" => s_box_out <= "10101011"; --(X"AB")
```

```
            WHEN "00001111" => s_box_out <= "01110110"; --(X"76")
            --second row
WHEN "00010000" => s_box_out <= "11001010"; --(X"CA")
            WHEN "00010001" => s_box_out <= "10000010"; --(X"82")
            WHEN "00010010" => s_box_out <= "11001001"; --(X"C9")
            WHEN "00010011" => s_box_out <= "01111101"; --(X"7D")
            WHEN "00010100" => s_box_out <= "11111010"; --(X"FA")
            WHEN "00010101" => s_box_out <= "01011001"; --(X"59")
            WHEN "00010110" => s_box_out <= "01000111"; --(X"47")
            WHEN "00010111" => s_box_out <= "11110000"; --(X"F0")
            WHEN "00011000" => s_box_out <= "10101101"; --(X"AD")
            WHEN "00011001" => s_box_out <= "11010100"; --(X"D4")
            WHEN "00011010" => s_box_out <= "10100010"; --(X"A2")
            WHEN "00011011" => s_box_out <= "10101111"; --(X"AF")
            WHEN "00011100" => s_box_out <= "10011100"; --(X"9C")
            WHEN "00011101" => s_box_out <= "10100100"; --(X"A4")
            WHEN "00011110" => s_box_out <= "01110010"; --(X"72")
            WHEN "00011111" => s_box_out <= "11000000"; --(X"C0")
            --third row
WHEN "00100000" => s_box_out <= "10110111"; --(X"B7")
            WHEN "00100001" => s_box_out <= "11111101"; --(X"FD")
            WHEN "00100010" => s_box_out <= "10010011"; --(X"93")
            WHEN "00100011" => s_box_out <= "00100110"; --(X"26")
            WHEN "00100100" => s_box_out <= "00110110"; --(X"36")
            WHEN "00100101" => s_box_out <= "00111111"; --(X"3F")
            WHEN "00100110" => s_box_out <= "11110111"; --(X"F7")
            WHEN "00100111" => s_box_out <= "11001100"; --(X"CC")
            WHEN "00101000" => s_box_out <= "00110100"; --(X"34")
            WHEN "00101001" => s_box_out <= "10100101"; --(X"A5")
            WHEN "00101010" => s_box_out <= "11100101"; --(X"E5")
            WHEN "00101011" => s_box_out <= "11110001"; --(X"F1")
            WHEN "00101100" => s_box_out <= "01110001"; --(X"71")
            WHEN "00101101" => s_box_out <= "11011000"; --(X"D8")
            WHEN "00101110" => s_box_out <= "00110001"; --(X"31")
            WHEN "00101111" => s_box_out <= "00010101"; --(X"15")
            --forth row
            WHEN "00110000" => s_box_out <= "00000100"; --(X"04")
            WHEN "00110001" => s_box_out <= "11000111"; --(X"C7")
            WHEN "00110010" => s_box_out <= "00100011"; --(X"23")
            WHEN "00110011" => s_box_out <= "11000011"; --(X"C3")
            WHEN "00110100" => s_box_out <= "00011000"; --(X"18")
            WHEN "00110101" => s_box_out <= "10010110"; --(X"96")
            WHEN "00110110" => s_box_out <= "00000101"; --(X"05")
            WHEN "00110111" => s_box_out <= "10011010"; --(X"9A")
```

```
WHEN "00111000" => s_box_out <= "00000111"; --(X"07")
WHEN "00111001" => s_box_out <= "00010010"; --(X"12")
WHEN "00111010" => s_box_out <= "10000000"; --(X"80")
WHEN "00111011" => s_box_out <= "11100010"; --(X"E2")
WHEN "00111100" => s_box_out <= "11101011"; --(X"EB")
WHEN "00111101" => s_box_out <= "00100111"; --(X"27")
WHEN "00111110" => s_box_out <= "10110010"; --(X"B2")
WHEN "00111111" => s_box_out <= "01110101"; --(X"75")
--fifth row
WHEN "01000000" => s_box_out <= "00001001"; --(X"09")
WHEN "01000001" => s_box_out <= "10000011"; --(X"83")
WHEN "01000010" => s_box_out <= "00101100"; --(X"2C")
WHEN "01000011" => s_box_out <= "00011010"; --(X"1A")
WHEN "01000100" => s_box_out <= "00011011"; --(X"1B")
WHEN "01000101" => s_box_out <= "01101110"; --(X"6E")
WHEN "01000110" => s_box_out <= "01011010"; --(X"5A")
WHEN "01000111" => s_box_out <= "10100000"; --(X"A0")
WHEN "01001000" => s_box_out <= "01010010"; --(X"52")
WHEN "01001001" => s_box_out <= "00111011"; --(X"3B")
WHEN "01001010" => s_box_out <= "11010110"; --(X"D6")
WHEN "01001011" => s_box_out <= "10110011"; --(X"B3")
WHEN "01001100" => s_box_out <= "00101001"; --(X"29")
WHEN "01001101" => s_box_out <= "11100011"; --(X"E3")
WHEN "01001110" => s_box_out <= "00101111"; --(X"2F")
WHEN "01001111" => s_box_out <= "10000100"; --(X"84")
--sixth row
WHEN "01010000" => s_box_out <= "01010011"; --(X"53")
WHEN "01010001" => s_box_out <= "11010001"; --(X"D1")
WHEN "01010010" => s_box_out <= "00000000"; --(X"00")
WHEN "01010011" => s_box_out <= "11101101"; --(X"ED")
WHEN "01010100" => s_box_out <= "00100000"; --(X"20")
WHEN "01010101" => s_box_out <= "11111100"; --(X"FC")
WHEN "01010110" => s_box_out <= "10110001"; --(X"B1")
WHEN "01010111" => s_box_out <= "01011011"; --(X"5B")
WHEN "01011000" => s_box_out <= "01101010"; --(X"6A")
WHEN "01011001" => s_box_out <= "11001011"; --(X"CB")
WHEN "01011010" => s_box_out <= "10111110"; --(X"BE")
WHEN "01011011" => s_box_out <= "00111001"; --(X"39")
WHEN "01011100" => s_box_out <= "01001010"; --(X"4A")
WHEN "01011101" => s_box_out <= "01001100"; --(X"4C")
WHEN "01011110" => s_box_out <= "01011000"; --(X"58")
WHEN "01011111" => s_box_out <= "11001111"; --(X"CF")
--seventh row
WHEN "01100000" => s_box_out <= "11010000"; --(X"D0")
```

```
WHEN "01100001" => s_box_out <= "11101111"; --(X"EF")
WHEN "01100010" => s_box_out <= "10101010"; --(X"AA")
WHEN "01100011" => s_box_out <= "11111011"; --(X"FB")
WHEN "01100100" => s_box_out <= "01000011"; --(X"43")
WHEN "01100101" => s_box_out <= "01001101"; --(X"4D")
WHEN "01100110" => s_box_out <= "00110011"; --(X"33")
WHEN "01100111" => s_box_out <= "10000101"; --(X"85")
WHEN "01101000" => s_box_out <= "01000101"; --(X"45")
WHEN "01101001" => s_box_out <= "11111001"; --(X"F9")
WHEN "01101010" => s_box_out <= "00000010"; --(X"02")
WHEN "01101011" => s_box_out <= "01111111"; --(X"7F")
WHEN "01101100" => s_box_out <= "01010000"; --(X"50")
WHEN "01101101" => s_box_out <= "00111100"; --(X"3C")
WHEN "01101110" => s_box_out <= "10011111"; --(X"9F")
WHEN "01101111" => s_box_out <= "10101000"; --(X"A8")
--eighth row
WHEN "01110000" => s_box_out <= "01010001"; --(X"51")
WHEN "01110001" => s_box_out <= "10100011"; --(X"A3")
WHEN "01110010" => s_box_out <= "01000000"; --(X"40")
WHEN "01110011" => s_box_out <= "10001111"; --(X"8F")
WHEN "01110100" => s_box_out <= "10010010"; --(X"92")
WHEN "01110101" => s_box_out <= "10011101"; --(X"9D")
WHEN "01110110" => s_box_out <= "00111000"; --(X"38")
WHEN "01110111" => s_box_out <= "11110101"; --(X"F5")
WHEN "01111000" => s_box_out <= "10111100"; --(X"BC")
WHEN "01111001" => s_box_out <= "10110110"; --(X"B6")
WHEN "01111010" => s_box_out <= "11011010"; --(X"DA")
WHEN "01111011" => s_box_out <= "00100001"; --(X"21")
WHEN "01111100" => s_box_out <= "00010000"; --(X"10")
WHEN "01111101" => s_box_out <= "11111111"; --(X"FF")
WHEN "01111110" => s_box_out <= "11110011"; --(X"F3")
WHEN "01111111" => s_box_out <= "11010010"; --(X"D2")
--ninth row
WHEN "10000000" => s_box_out <= "11001101"; --(X"CD")
WHEN "10000001" => s_box_out <= "00001100"; --(X"0C")
WHEN "10000010" => s_box_out <= "00010011"; --(X"13")
WHEN "10000011" => s_box_out <= "11101100"; --(X"EC")
WHEN "10000100" => s_box_out <= "01011111"; --(X"5F")
WHEN "10000101" => s_box_out <= "10010111"; --(X"97")
WHEN "10000110" => s_box_out <= "01000100"; --(X"44")
WHEN "10000111" => s_box_out <= "00010111"; --(X"17")
WHEN "10001000" => s_box_out <= "11000100"; --(X"C4")
WHEN "10001001" => s_box_out <= "10100111"; --(X"A7")
WHEN "10001010" => s_box_out <= "01111110"; --(X"7E")
```

```
WHEN "10001011" => s_box_out <= "00111101"; --(X"3D")
WHEN "10001100" => s_box_out <= "01100100"; --(X"64")
WHEN "10001101" => s_box_out <= "01011101"; --(X"5D")
WHEN "10001110" => s_box_out <= "00011001"; --(X"19")
WHEN "10001111" => s_box_out <= "01110011"; --(X"73")
--tenth row
WHEN "10010000" => s_box_out <= "01100000"; --(X"60")
WHEN "10010001" => s_box_out <= "10000001"; --(X"81")
WHEN "10010010" => s_box_out <= "01001111"; --(X"4F")
WHEN "10010011" => s_box_out <= "11011100"; --(X"DC")
WHEN "10010100" => s_box_out <= "00100010"; --(X"22")
WHEN "10010101" => s_box_out <= "00101010"; --(X"2A")
WHEN "10010110" => s_box_out <= "10010000"; --(X"90")
WHEN "10010111" => s_box_out <= "10001000"; --(X"88")
WHEN "10011000" => s_box_out <= "01000110"; --(X"46")
WHEN "10011001" => s_box_out <= "11101110"; --(X"EE")
WHEN "10011010" => s_box_out <= "10111000"; --(X"B8")
WHEN "10011011" => s_box_out <= "00010100"; --(X"14")
WHEN "10011100" => s_box_out <= "11011110"; --(X"DE")
WHEN "10011101" => s_box_out <= "01011110"; --(X"5E")
WHEN "10011110" => s_box_out <= "00001011"; --(X"0B")
WHEN "10011111" => s_box_out <= "11011011"; --(X"DB")
--eleventh row
WHEN "10100000" => s_box_out <= "11100000"; --(X"E0")
WHEN "10100001" => s_box_out <= "00110010"; --(X"32")
WHEN "10100010" => s_box_out <= "00111010"; --(X"3A")
WHEN "10100011" => s_box_out <= "00001010"; --(X"0A")
WHEN "10100100" => s_box_out <= "01001001"; --(X"49")
WHEN "10100101" => s_box_out <= "00000110"; --(X"06")
WHEN "10100110" => s_box_out <= "00100100"; --(X"24")
WHEN "10100111" => s_box_out <= "01011100"; --(X"5C")
WHEN "10101000" => s_box_out <= "11000010"; --(X"C2")
WHEN "10101001" => s_box_out <= "11010011"; --(X"D3")
WHEN "10101010" => s_box_out <= "10101100"; --(X"AC")
WHEN "10101011" => s_box_out <= "01100010"; --(X"62")
WHEN "10101100" => s_box_out <= "10010001"; --(X"91")
WHEN "10101101" => s_box_out <= "10010101"; --(X"95")
WHEN "10101110" => s_box_out <= "11100100"; --(X"E4")
WHEN "10101111" => s_box_out <= "01111001"; --(X"79")
--twelveth row
WHEN "10110000" => s_box_out <= "11100111"; --(X"E7")
WHEN "10110001" => s_box_out <= "11001000"; --(X"C8")
WHEN "10110010" => s_box_out <= "00110111"; --(X"37")
WHEN "10110011" => s_box_out <= "01101101"; --(X"6D")
```

```
WHEN "10110100" => s_box_out <= "10001101"; --(X"8D")
WHEN "10110101" => s_box_out <= "11010101"; --(X"D5")
WHEN "10110110" => s_box_out <= "01001110"; --(X"4E")
WHEN "10110111" => s_box_out <= "10101001"; --(X"A9")
WHEN "10111000" => s_box_out <= "01101100"; --(X"6C")
WHEN "10111001" => s_box_out <= "01010110"; --(X"56")
WHEN "10111010" => s_box_out <= "11110100"; --(X"F4")
WHEN "10111011" => s_box_out <= "11101010"; --(X"EA")
WHEN "10111100" => s_box_out <= "01100101"; --(X"65")
WHEN "10111101" => s_box_out <= "01111010"; --(X"7A")
WHEN "10111110" => s_box_out <= "10101110"; --(X"AE")
WHEN "10111111" => s_box_out <= "00001000"; --(X"08")
--thirteenth row
WHEN "11000000" => s_box_out <= "10111010"; --(X"BA")
WHEN "11000001" => s_box_out <= "01111000"; --(X"78")
WHEN "11000010" => s_box_out <= "00100101"; --(X"25")
WHEN "11000011" => s_box_out <= "00101110"; --(X"2E")
WHEN "11000100" => s_box_out <= "00011100"; --(X"1C")
WHEN "11000101" => s_box_out <= "10100110"; --(X"A6")
WHEN "11000110" => s_box_out <= "10110100"; --(X"B4")
WHEN "11000111" => s_box_out <= "11000110"; --(X"C6")
WHEN "11001000" => s_box_out <= "11101000"; --(X"E8")
WHEN "11001001" => s_box_out <= "11011101"; --(X"DD")
WHEN "11001010" => s_box_out <= "01110100"; --(X"74")
WHEN "11001011" => s_box_out <= "00011111"; --(X"1F")
WHEN "11001100" => s_box_out <= "01001011"; --(X"4B")
WHEN "11001101" => s_box_out <= "10111101"; --(X"BD")
WHEN "11001110" => s_box_out <= "10001011"; --(X"8B")
WHEN "11001111" => s_box_out <= "10001010"; --(X"8A")
--forteenth row
WHEN "11010000" => s_box_out <= "01110000"; --(X"70")
WHEN "11010001" => s_box_out <= "00111110"; --(X"3E")
WHEN "11010010" => s_box_out <= "10110101"; --(X"B5")
WHEN "11010011" => s_box_out <= "01100110"; --(X"66")
WHEN "11010100" => s_box_out <= "01001000"; --(X"48")
WHEN "11010101" => s_box_out <= "00000011"; --(X"03")
WHEN "11010110" => s_box_out <= "11110110"; --(X"F6")
WHEN "11010111" => s_box_out <= "00001110"; --(X"0E")
WHEN "11011000" => s_box_out <= "01100001"; --(X"61")
WHEN "11011001" => s_box_out <= "00110101"; --(X"35")
WHEN "11011010" => s_box_out <= "01010111"; --(X"57")
WHEN "11011011" => s_box_out <= "10111001"; --(X"B9")
WHEN "11011100" => s_box_out <= "10000110"; --(X"86")
WHEN "11011101" => s_box_out <= "11000001"; --(X"C1")
```

```
                WHEN "11011110" => s_box_out <= "00011101"; --(X"1D")
                WHEN "11011111" => s_box_out <= "10011110"; --(X"9E")
                --fifteenth row
                WHEN "11100000" => s_box_out <= "11100001"; --(X"E1")
                WHEN "11100001" => s_box_out <= "11111000"; --(X"F8")
                WHEN "11100010" => s_box_out <= "10011000"; --(X"98")
                WHEN "11100011" => s_box_out <= "00010001"; --(X"11")
                WHEN "11100100" => s_box_out <= "01101001"; --(X"69")
                WHEN "11100101" => s_box_out <= "11011001"; --(X"D9")
                WHEN "11100110" => s_box_out <= "10001110"; --(X"8E")
                WHEN "11100111" => s_box_out <= "10010100"; --(X"94")
                WHEN "11101000" => s_box_out <= "10011011"; --(X"9B")
                WHEN "11101001" => s_box_out <= "00011110"; --(X"1E")
                WHEN "11101010" => s_box_out <= "10000111"; --(X"87")
                WHEN "11101011" => s_box_out <= "11101001"; --(X"E9")
                WHEN "11101100" => s_box_out <= "11001110"; --(X"CE")
                WHEN "11101101" => s_box_out <= "01010101"; --(X"55")
                WHEN "11101110" => s_box_out <= "00101000"; --(X"28")
                WHEN "11101111" => s_box_out <= "11011111"; --(X"DF")
                --sixteenth row
                WHEN "11110000" => s_box_out <= "10001100"; --(X"8C")
                WHEN "11110001" => s_box_out <= "10100001"; --(X"A1")
                WHEN "11110010" => s_box_out <= "10001001"; --(X"89")
                WHEN "11110011" => s_box_out <= "00001101"; --(X"0D")
                WHEN "11110100" => s_box_out <= "10111111"; --(X"BF")
                WHEN "11110101" => s_box_out <= "11100110"; --(X"E6")
                WHEN "11110110" => s_box_out <= "01000010"; --(X"42")
                WHEN "11110111" => s_box_out <= "01101000"; --(X"68")
                WHEN "11111000" => s_box_out <= "01000001"; --(X"41")
                WHEN "11111001" => s_box_out <= "10011001"; --(X"99")
                WHEN "11111010" => s_box_out <= "00101101"; --(X"2D")
                WHEN "11111011" => s_box_out <= "00001111"; --(X"0F")
                WHEN "11111100" => s_box_out <= "10110000"; --(X"B0")
                WHEN "11111101" => s_box_out <= "01010100"; --(X"54")
                WHEN "11111110" => s_box_out <= "10111011"; --(X"BB")
                WHEN "11111111" => s_box_out <= "00010110"; --(X"16")

                WHEN OTHERS => s_box_out <= "XXXXXXXX" ;
                END CASE;
  END PROCESS;
END behavioral;
```

Package my_package

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------

PACKAGE my_package IS
  TYPE row IS ARRAY (15 DOWNTO 0) OF std_logic_vector(7 DOWNTO 0);
  TYPE matrix IS ARRAY (10 DOWNTO 0) OF row;
  TYPE key_matrix IS ARRAY (10 DOWNTO 0) OF std_logic_vector(127
DOWNTO 0);
  TYPE key_matrix2 IS ARRAY (1 TO 10) OF std_logic_vector(127 DOWNTO 0);
  TYPE key_matrix3 IS ARRAY (1 TO 9) OF std_logic_vector(127 DOWNTO 0);
  TYPE round_array IS ARRAY (10 DOWNTO 1) OF std_logic_vector(7 DOWNTO
0);
  TYPE mix_array IS ARRAY (15 DOWNTO 0) OF std_logic_vector(8 DOWNTO
0);
END my_package;
----------------------------------

PACKAGE BODY my_package IS

END my_package;
```

ΠΑΡΑΡΤΗΜΑ 2
ΕΝΤΟΛΕΣ dofiles

1^{ο} Σετ Εισόδων

add wave /milenage/*

force- freeze /K
01000110010110110101110011101000101100011001100110110100100111111010101001011110000101000101110111000100011100010100110101111000

force -freeze /RAND
00100011010101010011110010111110100101100011011110101000100111010010000110001010111001100100110110101110010001111011111100110101 0

force -freeze /SQN 111111111001101110110100110100001011011000000111 0

force -freeze /AMF 1011100110111001 0

run 2000

2^{ο} Σετ Εισόδων

add wave /milenage/*

force- freeze /K 9e5944ae a94b8116 5c82fbf9 f32db751 0

force -freeze /RAND ce83dbc5 4ac0274a 157c17f8 0d017bd6

force -freeze /SQN 0b604a81 eca8

force -freeze /AMF 9e09

run 2000

ΒΙΒΛΙΟΓΡΑΦΙΑ

- "Σχεδιασμός κυκλωμάτων με την VHDL", Volnei A. Pedroni.
- "Ψηφιακή Σχεδίαση Αρχές & Πρακτικές", John F. Wakerly.
- "LTE Security", Dan Forsberg – Gunther Horn – Wolf – Dietrich Moeller – Valtteri Niemi.
- "Ανάπτυξη πλήρους ενσωματωμένου συστήματος, βασισμένου σε πλατφόρμα επεξεργαστή - FPGA με λειτουργικό σύστημα Linux για εκτέλεση κρυπτογραφικών αλγορίθμων SHA -512 και AES", Σταύρος Αντωνόπουλος – Νικολετάκης.
- "A VHDL implementation of the Advanced Encryption Standard-Rijndael Algorithm", Rajender Manteena, 2004
- "Μελέτη της αρχιτεκτονικής των ολοκληρωμένων κυκλωμάτων FPGAs και CPLDs και εφαρμογές", Πτυχιακή εργασία, Προύντζου Χρυσή, ΑΤΕΙ Σερρών, Σχολή Τεχνολογικών Εφαρμογών, Τμήμα Πληροφορικής & Επικοινωνιών 2008
- "Υλοποίηση ενός μικροεπεξεργαστή με VHDL κώδικα", Πτυχιακή Εργασία, Ντούσκα Ελπίδα, ΤΕΙ Ηπείρου, Τμήμα Τεχνολογίας Πληροφορικής και Τηλεπικοινωνιών.
- ESTI SAGE Task Force for 3GPP Authentication Function Algorithms. Specification of The MILENAGE Algorithm Set: an Example Algorithm Set for 3GPP Authentication and Key Generation Function f1, f1*, f2, f3, f4, f5 και f5*, November 2000
  - Document 1: Algorithm Specification
  - Document 2: Implementers' Test Data
  - Document 3: Design Conformance Test Data

ΔΙΑΔΥΚΤΙΑΚΕΣ ΠΗΓΕΣ

- http://www.etsi.org
- https://en.wikipedia.org/wiki/4G
- https://en.wikipedia.org/wiki/3G
- https://en.wikipedia.org/wiki/MIMO-OFDM
- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
- http://www.formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng.swf
- http://www.vhdl-online.de/
- https://el.wikipedia.org/wiki/FPGA
- http://www.xilinx.com
- http://www.mpf.org.in/pdf/technology/SIM_Card_Committee_Report.pdf
- https://www.blackhat.com/docs/us-15/materials/us-15-Yu-Cloning-3G-4G-SIM-Cards-With-A-PC-And-An-Oscilloscope-Lessons-Learned-In-Physical-Security-wp.pdf
- http://web.csulb.edu/projects/npu/Rijndael_AES/Rinjdael_AES.htm
- http://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html