ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση σε αναδιατασσόμενη λογική των πινάκων ουράνιου τόξου (rainbow tables) για την αποκρυπτογράφηση του κρυπτογραφικού αλγόριθμου A5/3 που χρησιμοποιείται σε δίκτυα 3^{ης} γενιάς (3G)

Εξεταστική επιτροπή

Παπαευσταθίου Ιωάννης (επιβλέπων)	Επίκουρος καθηγητής
Δόλλας Απόστολος	Καθηγητής
Μανιφάβας Χαράλαμπος	Επίκουρος καθηγητής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ ΤΟΥ ΦΟΙΤΗΤΗ

ΠΑΝΑΓΙΩΤΗ ΠΑΠΑΝΤΩΝΑΚΗ

Αφιερωμένο στην οικογένειά και στους φίλους μου.

Πίνακας περιεχομένων

Λίστα Εικόνων-Πινάκων

1 Εισαγωγή

Η κρυπτογραφία είναι ένας κλάδος της επιστήμης της κρυπτολογίας που ασχολείται με την μελέτη της ασφαλούς επικοινωνίας. Η κρυπτογράφηση δημόσιου κλειδιού με την οποία θα ασχοληθούμε περισσότερο σε αυτή την εργασία, εφαρμόζεται με αλγόριθμους που χρησιμοποιούν ένα δημόσιο κλειδί έτσι ώστε να εξασφαλίσουν την εμπιστευτικότητα και την επικύρωση των δεδομένων.

Η κρυπτανάλυση είναι η μέθοδος με την οποία προσπαθούμε να ανακτήσουμε έναν κωδικό ώστε να μπορέσουμε να καταλάβουμε το περιεχόμενο ενός κρυπτογραφημένου μηνύματος. Η διαδικασία της κρυπτανάλυσης χρησιμοποιείται κυρίως για ερευνητικούς λόγους έτσι ώστε να διαπιστωθούν αδυναμίες των αλγόριθμων κρυπτογράφησης που υπάρχουν ήδη.

Υπάρχουν πολλές μέθοδοι κρυπτανάλυσης όπως η brute force attack, plaintext attack κλπ αλλά σε αυτή την εργασία θα ασχοληθούμε με τους πίνακες ουράνιου τόξου (rainbow tables). Οι πίνακες ουράνιου τόξου είναι μια μέθοδος που χρησιμοποιεί προϋπολογισμένους πίνακες ώστε να επιτύχει την ανάκτηση ενός κλειδιού από ένα κρυπτογραφημένο κείμενο.

Η κινητή τηλεφωνία είναι ένας τομέας στον οποίο είναι απαραίτητη η κρυπτογραφία καθώς διαφορετικά θα καθιστούσε την διαδικασία της υποκλοπής πολύ εύκολη. Εδώ και πολλά χρόνια δημοσιεύονται καινούριοι αλγόριθμοι οι οποίοι προσπαθούν να παρέχουν την απαιτούμενη ασφάλεια στις συνομιλίες. Σε αυτή την εργασία θα ασχοληθούμε με την εφαρμογή των rainbow tables στον κρυπτογραφικό αλγόριθμο A5/3 ο οποίος χρησιμοποιείται για την κρυπτογράφηση δεδομένων κινητής τηλεφωνίας τρίτης γενιάς (3G).

Στην εργασία έχουμε την παρακάτω δομή για την διάρθρωση των κεφαλαίων : πρώτα θα ορίσουμε κάποιες βασικές έννοιες σχετικές με την κρυπτογραφία και την κρυπτανάλυση. Έπειτα θα αναλύσουμε την δομή του κρυπτογραφικού αλγόριθμου A5/3, τις αδυναμίες του καθώς και την υλοποίηση που κάναμε σε hardware, τις βελτιστοποιήσεις που πραγματοποιήσαμε καθώς και τον έλεγχο ορθότητας της σχεδίασής μας. Μετά, θα ασχοληθούμε με το θεωρητικό υπόβαθρο των πινάκων ουράνιου τόξου, πως αυτοί μπορούν να μας βοηθήσουν στο πρόβλημά μας καθώς και την πρακτική τους εφαρμογή, δηλαδή την υλοποίησή τους σε hardware. Σκοπεύουμε να υλοποιήσουμε ένα ολοκληρωμένο σύστημα παραγωγής πινάκων αποκλειστικά σε hardware έτσι ώστε να μπορέσουμε να επιτύχουμε κάποια βελτίωση σε σχέση με τις αντίστοιχες software υλοποιήσεις οι οποίες εφαρμόζονται μέχρι τώρα. Τα αποτελέσματα αυτών των συγκρίσεων καθώς και μελλοντικές βελτιώσεις πάνω στην σχεδίασή μας αναλύονται στο τελευταίο κεφάλαιο.

2 Κρυπτογραφία - Κρυπτανάλυση

2.1 Γενικά για την κρυπτογραφία

Η κρυπτογραφία είναι η επιστήμη που ασχολείται με την μελέτη της μετατροπής μια πληροφορίας από την κανονική της, κατανοητή μορφή σε μια ακατάληπτη καθιστώντας την μη αναγνώσιμη με σκοπό την μεταφορά της διαμέσου ενός μέσου με ασφάλεια.

Ιστορικά, η κρυπτογραφία μέχρι τις αρχές του 1970, χρησιμοποιούνταν κυρίως για στρατιωτικές και κυβερνητικές εφαρμογές. Από το 1980 και μετά, οι εταιρείες τηλεπικοινωνιών ήταν αυτές που πρώτες δημιούργησαν συστήματα κρυπτογράφησης σχεδιασμένα για hardware. Η πρώτη μαζικής παραγωγής κρυπτογραφική εφαρμογή ήταν το πρώτο κινητό τηλέφωνο, στα τέλη της δεκαετίας του 1980. Σήμερα, ο καθένας μας χρησιμοποιεί εφαρμογές της κρυπτογραφίας καθημερινά. Τέτοια παραδείγματα είναι το ξεκλείδωμα ενός αυτοκινήτου με χρήση ηλεκτρονικού κλειδιού, η σύνδεση του υπολογιστή μας σε ένα δίκτυο ασύρματης δικτύωσης (wireless LAN), η αγορά αγαθών με χρήση πιστωτικής κάρτας, η πραγματοποίηση μιας τηλεφωνικής συνομιλίας είτε με το δίκτυο GSM είτε με voice-over-IP. Δεν υπάρχει αμφιβολία ότι στο μέλλον εφαρμογές όπως τα έξυπνα κτίρια θα καταστήσουν την κρυπτογραφία αναγκαία.

Η ανάπτυξη των υπολογιστικών συστημάτων μετά την εποχή του δευτέρου παγκοσμίου πολέμου κατέστησε δυνατή την δημιουργία πολυπλοκότερων κρυπταλγόριθμων. Οι υπολογιστές επέτρεψαν την κρυπτογράφηση οποιασδήποτε πληροφορίας η οποία ήταν σε δυαδική μορφή σε αντίθεση με τους κλασικούς κρυπταλγόριθμους που χρησιμοποιούνταν μέχρι τότε οι οποίοι αφορούσαν μόνο γραπτά και προφορικά κείμενα.

Ακαδημαϊκή έρευνα σχετικά με την κρυπτογραφία άρχισε να υπάρχει από το 1970 και μετά με την IBM να εκδίδει τον DES (data encryption standard) το 1976 και τους Rivest, Shamir και Adleman να εκδίδουν τον αλγόριθμο RSA το 1978. Επίσης, μεγάλη εξέλιξη για την κρυπτογραφία ήταν η εισαγωγή της έννοιας της κρυπτογράφησης δημόσιου κλειδιού που έγινε από τους Diffie και Hellman το 1976. Ο αλγόριθμος RSA ήταν ο πρώτος πρακτικά χρήσιμος αλγόριθμος που χρησιμοποιούσε κρυπτογράφηση δημόσιου κλειδιού και την έννοια της ψηφιακής υπογραφής. Από τότε η κρυπτογραφία άρχισε να γίνεται απαραίτητο εργαλείο για τις τηλεπικοινωνίες, τα δίκτυα υπολογιστών και την ασφάλεια των υπολογιστών γενικότερα.
Η χρήση της ψηφιακής υπογραφής άρχισε να γίνεται πιο έντονη και το πρώτο διεθνές standard για ψηφιακές υπογραφές υιοθετήθηκε το 1991 και ήταν βασισμένο στον αλγόριθμο RSA.

Η κρυπτογραφία επίσης εξελίσσεται παράλληλα με τις μεταβολές στην βιομηχανία των υπολογιστών. Για παράδειγμα, τα τελευταία χρόνια έχουν γίνει θεαματικές βελτιώσεις στην υπολογιστική ισχύ και κατά συνέπεια οι επιθέσεις ωμής βίας έμοιαζαν εφικτή απειλή για τους ήδη υπάρχοντες κρυπταλγόριθμους. Για τον λόγο αυτό, το μήκος των εμπιστευτικών κλειδιών που χρησιμοποιούνται έχει αυξηθεί έτσι ώστε να αποτρέπονται τέτοιους είδους επιθέσεις.

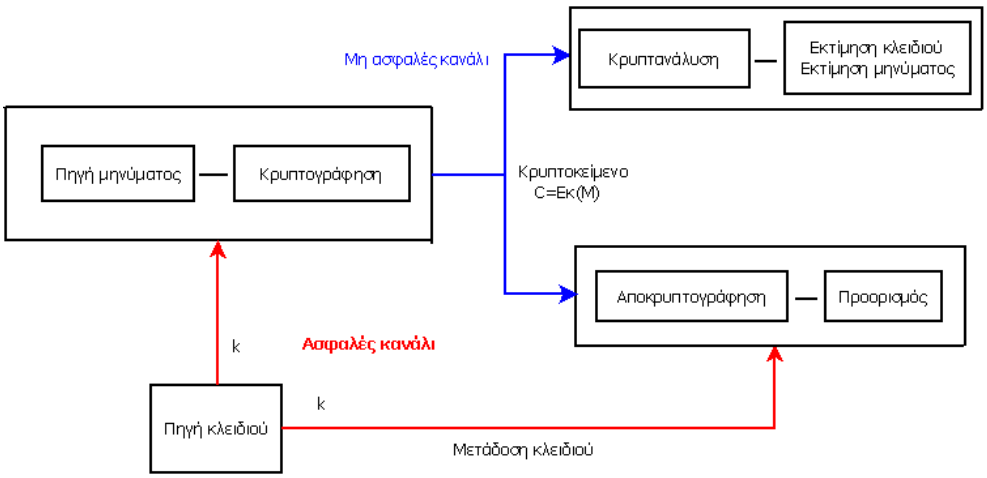
2.2 Περιγραφή γενικού συστήματος κρυπτογράφησης

Ο στόχος της κρυπτογραφίας είναι να δώσει την δυνατότητα σε δύο πρόσωπα να επικοινωνήσουν μέσα από ένα μη ασφαλές κανάλι (π.χ. στην περίπτωση της κινητής τηλεφωνίας τον αέρα), με τέτοιο τρόπο ώστε ένα τρίτο πρόσωπο, μη εξουσιοδοτημένο, να μην μπορεί να παρεμβληθεί στην επικοινωνία ή να κατανοήσει το περιεχόμενο των μηνυμάτων.

Ένα κρυπτοσύστημα αποτελείται από μια βασική πεντάδα παραμέτρων:

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

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



Εικόνα 2.1 Γενική μορφή συστήματος κρυπτογράφησης

Το σύστημα του σχήματος λειτουργεί με τον ακόλουθο τρόπο:

1. Ο αποστολέας επιλέγει ένα κλειδί συγκεκριμένου μήκους από τον χώρο των κλειδιών με τυχαίο τρόπο (ή τουλάχιστον όσο το δυνατόν πιο τυχαίο τρόπο) με τα στοιχεία του κλειδιού να ανήκουν σε ένα πεπερασμένο αλφάβητο.

2. Ο αποστολέας στέλνει το κλειδί στον παραλήπτη χρησιμοποιώντας ένα ασφαλές κανάλι επικοινωνίας.
3. Ο αποστολέας δημιουργεί το μήνυμα το οποίο θέλει να στείλει στον παραλήπτη από τον χώρο των μηνυμάτων.
4. Η συνάρτηση κρυπτογράφησης παίρνει τις δύο εισόδους (κλειδί και μήνυμα) και παράγει μια κρυπτοακολουθία συμβόλων (μη αναγνώσιμη) και η ακολουθία αυτή αποστέλλεται διαμέσου ενός μη ασφαλούς καναλιού.
5. Η συνάρτηση αποκρυπτογράφησης παίρνει ως όρισμα τις 2 τιμές (κλειδί και κρυπτογραφημένο κείμενο) και παράγει την ισοδύναμη ακολουθία μηνύματος.

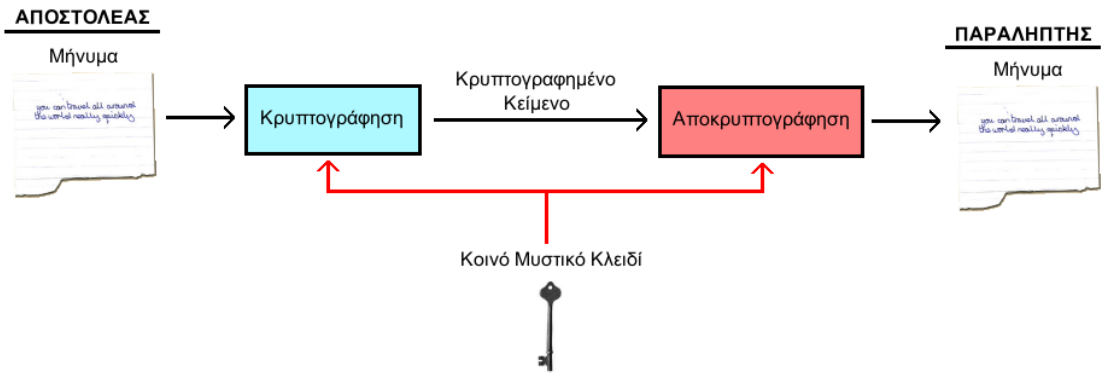
Ο πιθανός "επιτιθέμενος" στην συγκεκριμένη συνομιλία, μπορεί εύκολα να την παρακολουθήσει αλλά λόγω της μη γνώσης του κλειδιού κρυπτογράφησης δεν μπορεί να την μεταφράσει έτσι ώστε να την καταλάβει.

2.3 Τύποι αλγόριθμων κρυπτογράφησης

Υπάρχουν δύο βασικές κατηγορίες αλγορίθμων που χρησιμοποιούνται σήμερα για την κρυπτογραφία, οι συμμετρικοί και οι ασύμμετροι (ή δημόσιου κλειδιού).

Στους συμμετρικούς αλγόριθμους κρυπτογράφησης υπάρχει ένα κοινό μυστικό κλειδί και με βάση αυτό γίνεται η διαδικασία της κρυπτογράφησης και της αποκρυπτογράφησης. Οι μοναδικοί που επιτρέπεται να γνωρίζουν αυτό το κλειδί είναι ο αποστολέας και ο παραλήπτης. Το ίδιο κλειδί είναι αυτό που χρησιμοποιείται τόσο στην διαδικασία της κρυπτογράφησης όσο και στην διαδικασία της αποκρυπτογράφησης.

Ο πρώτος συμμετρικός αλγόριθμος ιστορικά είναι ο αλγόριθμος του Καίσαρα ο οποίος χρησιμοποιούσε μια απλή αντικατάσταση γραμμάτων. Γνωστοί σύγχρονοι συμμετρικοί αλγόριθμοι είναι ο DES (data encryption standard), AES (advanced encryption standard), RC5 (Rivest Cipher 5).

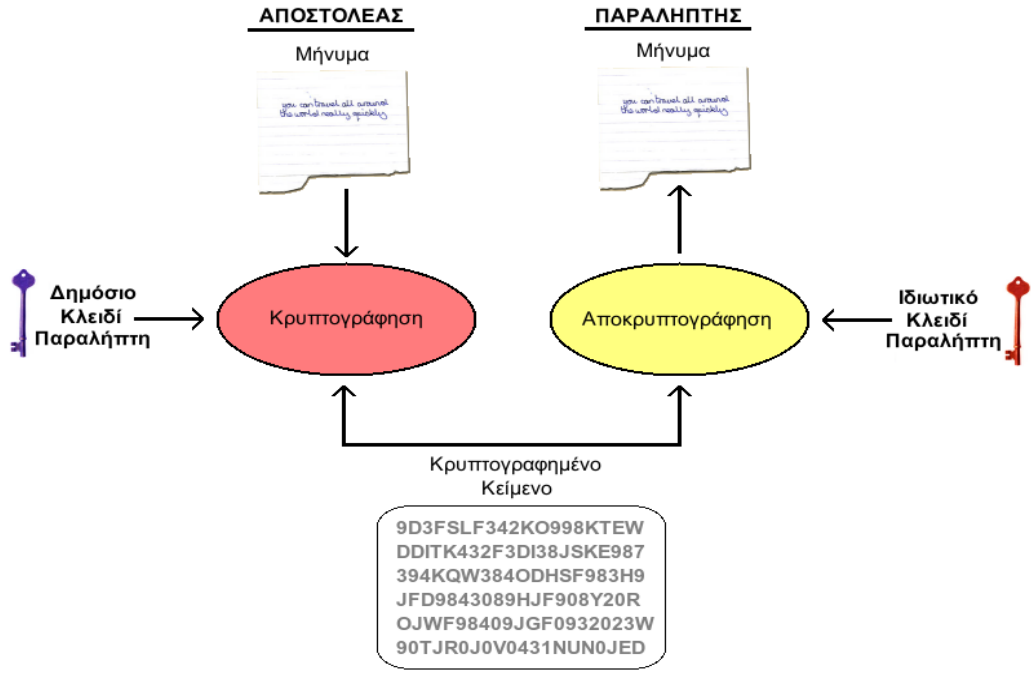


Εικόνα 2.2 Κρυπτογράφηση Δημόσιου Κλειδιού

Βασικό πλεονέκτημα της συμμετρικής κρυπτογράφησης είναι οι υψηλές ταχύτητες μετατροπής ενός μηνύματος σε κρυπτογραφημένο κείμενο καθώς και οι μικρές απαιτήσεις σε υπολογιστική ισχύ.

Η κύρια αδυναμία των συμμετρικών αλγόριθμων κρυπτογράφησης είναι ότι λαμβάνουν σαν δεδομένο ότι η μετάδοση του κοινού μυστικού κλειδιού είναι ασφαλής και δεν μπορεί να παρακολουθηθεί από τρίτους. Κάτι τέτοιο δεν είναι εφικτό σε όλα τα δίκτυα επικοινωνίας λόγω πρακτικών αλλά και λειτουργικών δυσκολιών καθώς θα πρέπει να υπάρχει ένα ασφαλές κανάλι επικοινωνίας. Η μεταφορά του κλειδιού γίνεται συνήθως μέσω μιας έμπιστης τρίτης οντότητας. Μια ακόμη αδυναμία αυτής της μεθόδου είναι ότι δεν είναι εύκολο υλοποιηθεί σε μεγαλύτερη κλίμακα. Όσο μεγαλώνει ο αριθμός των χρηστών που επικοινωνούν σε ένα δίκτυο, θα μεγαλώνει και ο αριθμός των κλειδιών που χρειάζεται να παραχθούν. Στο πρόβλημα αυτό θα πρέπει να προσμετρήσουμε επίσης και την ανάγκη περιοδικής αντικατάστασης των κλειδιών που παράγονται για λόγους ασφάλειας.

Στους ασύμμετρους αλγόριθμους (ή δημόσιου κλειδιού) κάθε ένας που συμμετέχει σε μια ανταλλαγή πληροφορίας έχει δύο κλειδιά, ένα δημόσιο και ένα ιδιωτικό. Το δημόσιο κλειδί είναι γνωστό σε όλους και χρησιμοποιείται από όποιον θέλει να στείλει ένα κρυπτογραφημένο μήνυμα στον ιδιοκτήτη του κλειδιού. Αυτό το δημόσιο κλειδί όμως, δεν μπορεί να χρησιμοποιηθεί για την αποκρυπτογράφηση του μηνύματος. Για να γίνει αυτή, θα πρέπει να χρησιμοποιηθεί το ιδιωτικό κλειδί το οποίο είναι γνωστό μόνο στον ιδιοκτήτη του. Τα δύο κλειδιά αν και συνδέονται μεταξύ τους με κάποια μαθηματική σχέση, είναι αρκετά διαφορετικά έτσι ώστε η γνώση του ενός να μην επιτρέπει τον υπολογισμό του άλλου. Η μέθοδος αυτή κρυπτογράφησης είχε προταθεί στα μέσα της δεκαετίας του ’70 από τους Diffie και Hellman.



Εικόνα 2.3 Αλγόριθμος Ασύμμετρης Κρυπτογράφησης

Βασική προϋπόθεση για έναν αλγόριθμο ασύμμετρης κρυπτογράφησης είναι ότι το ζεύγος των κλειδιών είναι συμπληρωματικό το ένα προς το άλλο, δηλαδή το κείμενο που κρυπτογραφείται με το δημόσιο κλειδί μπορεί να αποκρυπτογραφηθεί μόνο με ένα ιδιωτικό κλειδί και αντίστροφα. Ουσιαστικά το δημόσιο κλειδί είναι προσβάσιμο από πολλούς και χρησιμοποιείται για την πιστοποίηση ψηφιακών υπογραφών. Το ιδιωτικό κλειδί πρέπει να είναι προστατευμένο από τον ιδιοκτήτη του και χρησιμοποιείται για την ψηφιακή υπογραφή μηνυμάτων.

Το κύριο πλεονέκτημα της ασύμμετρης κρυπτογράφησης είναι ότι δεν απαιτείται ανταλλαγή μυστικού κλειδιού. Επίσης, δημιουργείται μια πολύ σημαντική λειτουργία στον τομέα της κρυπτογραφίας, η ψηφιακή υπογραφή μηνυμάτων. Παρ όλα αυτά, η ασύμμετρη κρυπτογράφηση έχει μεγάλη χρονική πολυπλοκότητα και μεγάλες απαιτήσεις σε υπολογιστική ισχύ. Για τον λόγο αυτό δεν προτιμάται σε μεταδόσεις που μας ενδιαφέρει η ταχύτητα στην μεταφορά των δεδομένων (π.χ. στην κινητή τηλεφωνία).

Μια κατηγορία αλγορίθμων κρυπτογράφησης είναι επίσης και τα κρυπτογραφικά πρωτόκολλα (cryptographic protocols). Στην περίπτωση των πρωτοκόλλων αυτών, οι συμμετρικοί και ασύμμετροι αλγόριθμοι αντιμετωπίζονται ως κομμάτια μιας γενικότερης δομής στην οποία χτίζονται ασφαλείς εφαρμογές. Ένα τέτοιο παράδειγμα είναι το πρωτόκολλο TLS (Transport Layer Security) το οποίο χρησιμοποιείται για να διασφαλίσει την επικοινωνία πελάτη-εξυπηρετητή στο διαδίκτυο.

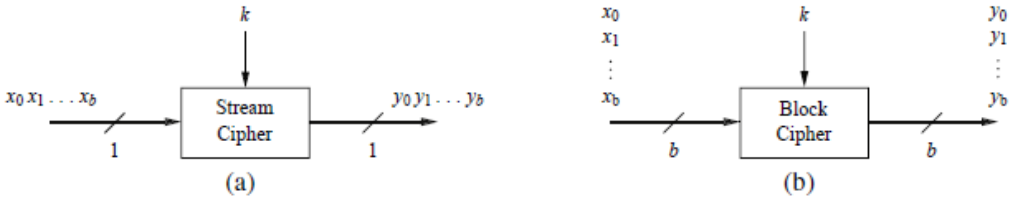
Ακόμα, θα μπορούσαμε να κατηγοριοποιήσουμε τους αλγόριθμους κρυπτογράφησης σε αλγόριθμους δέσμης και αλγόριθμους ροής. Οι κρυπταλγόριθμοι δέσμης τεμαχίζουν σε τμήματα το αρχικό κείμενο και κρυπτογραφούν το κάθε τμήμα ξεχωριστά. Για την κρυπτογράφηση αυτή χρησιμοποιείται το μυστικό κλειδί. Στους αλγόριθμους αυτούς θα πρέπει να επιλέγεται αρκετά μεγάλο μέγεθος τμήματος έτσι ώστε να αποτρέπονται επιθέσεις λεξικού (dictionary attacks). Οι κρυπταλγόριθμοι ροής χρησιμοποιούνται για την κρυπτογράφηση μιας συνεχούς ροής δεδομένων (data stream). Για την κρυπτογράφηση αρχικά επιλέγεται μια γεννήτρια παραγωγής κλειδιών η οποία δέχεται σαν είσοδο ένα μυστικό κλειδί και παράγει μια τυχαία ακολουθία από bits, το key stream.

2.4 Τύποι αλγορίθμων συμμετρικής κρυπτογράφησης

Οι αλγόριθμοι συμμετρικής κρυπτογράφησης, όπως είναι και ο αλγόριθμος που θα μας απασχολήσει σε αυτή την εργασία, χωρίζονται σε δύο βασικές κατηγορίες, τα block ciphers και τα stream ciphers.

Τα stream ciphers κρυπτογραφούν το κάθε bit ξεχωριστά. Αυτό επιτυγχάνεται με την επιλογή ενός bit από ένα key stream και ενός bit πληροφορίας, όπως φαίνεται και από την εικόνα 1.4(a). Υπάρχουν διαφορετικές υλοποιήσεις για τα stream ciphers, υλοποιήσεις όπου το κλειδί εξαρτάται από το κρυπτογραφημένο κείμενο αλλά και υλοποιήσεις όπου το κλειδί είναι ανεξάρτητο.

Τα block ciphers (εικόνα 1.4b), κρυπτογραφούν ένα ολόκληρο block από bits χρησιμοποιώντας για όλο το block το ίδιο κλειδί. Τα block ciphers έχουν συγκεκριμένο μέγεθος block, συνήθως 64 (όπως ο DES, data encryption standard) ή 128 bits (όπως το AES, advanced encryption standard). Ο αλγόριθμος A5/3 με τον οποίο θα ασχοληθούμε είναι περίπτωση block cipher.



Εικόνα 2.4 Διαδικασία κρυπτογράφησης για stream και block ciphers

2.5 Προϋποθέσεις κρυπτογραφικού αλγόριθμου

Για να είναι ασφαλής ο κρυπταλγόριθμος ροής θα πρέπει να πληρούνται οι παρακάτω προϋποθέσεις :

Το key stream που παράγεται θα πρέπει να είναι όσο το δυνατόν πιο τυχαίο. Υπάρχουν ειδικές μέθοδοι που χρησιμοποιούνται για την δοκιμή καταλληλότητας των key stream που παράγονται.

Το key stream θα πρέπει επίσης να έχει μεγάλη γραμμική ισοδυναμία (linear equivalence). Οποιαδήποτε ακολουθία μπορεί να παραχθεί χρησιμοποιώντας γραμμικές μεθόδους, για παράδειγμα μπορούμε να υπολογίσουμε την τιμή μιας ακολουθίας με βάση τις τιμές της προηγούμενής της. Αν στον υπολογισμό αυτό χρησιμοποιηθεί ένας μεγάλος αριθμός από προηγούμενες ακολουθίες τότε έχουμε μεγάλη γραμμική ισοδυναμία. Γενικά, η μεγάλη γραμμική ισοδυναμία βοηθάει ώστε ο αλγόριθμος να έχει μεγαλύτερη ασφάλεια.

Οι τέσσερις βασικές λειτουργίες τις οποίες πρέπει να παρέχει η κρυπτογραφία είναι οι εξής:

- *Εμπιστευτικότητα* : Είναι η υπηρεσία η οποία θα διασφαλίσει να κρατήσει κρυφή την προς μετάδοση πληροφορία από οποιονδήποτε δεν είναι εξουσιοδοτημένος να την διαβάσει. Υπάρχουν πολλές διαφορετικές προσεγγίσεις για την παροχή εμπιστευτικότητας σε μια επικοινωνία, από φυσική προστασία μέχρι την χρήση μαθηματικών αλγορίθμων.
- *Ακεραιότητα* : Με την υπηρεσία αυτή φροντίζουμε να μην υπάρχει αλλαγή των δεδομένων που στέλνονται από κάποιον τρίτο. Ακόμα και αν υπάρχει κάποια τέτοια αλλαγή, θα πρέπει η συγκεκριμένη υπηρεσία να μπορεί να εντοπίσει ποιος πραγματοποίησε αυτή την αλλαγή. Η αλλαγή των δεδομένων περιλαμβάνει περιπτώσεις όπως εισαγωγή, διαγραφή ή αντικατάσταση τους.

- *Μη απάρνηση*: Ο αποστολέας ή ο παραλήπτης δεν μπορούν να αρνηθούν ή να ακυρώσουν προηγούμενες ενέργειές τους. Για παράδειγμα, ένας χρήστης μπορεί να έχει επικυρώσει την αγορά ενός προϊόντος και μετά ο ίδιος να αρνηθεί ότι πραγματοποίησε κάτι τέτοιο. Μια διαδικασία η οποία περιλαμβάνει έναν τρίτο ο οποίος θα αναλαμβάνει τέτοιες περιπτώσεις είναι αναγκαία.
- *Πιστοποίηση*: Δύο χρήστες που ξεκινούν μια επικοινωνία θα πρέπει να μπορούν να αναγνωρίσουν ο ένας τον άλλον και να πιστοποιήσουν τις ταυτότητές τους. Οι πληροφορίες οι οποίες στέλνονται μεταξύ τους θα πρέπει να εξετάζονται ως προς την αυθεντικότητα του αποστολέα, της ώρας αποστολής και του περιεχομένου των δεδομένων.

Ο βασικός στόχος της κρυπτογραφίας είναι να υλοποιεί τους παραπάνω τέσσερις τομείς τόσο θεωρητικά όσο και στην πράξη. Για παράδειγμα, στην περίπτωση της κρυπτογράφησης δημόσιου κλειδιού, για να επιτευχθεί η εμπιστευτικότητα ο αποστολέας θα πρέπει να χρησιμοποιήσει το δημόσιο κλειδί του παραλήπτη για να κρυπτογραφήσει το μήνυμα. Στη συνέχεια στέλνει το κρυπτογραφημένο μήνυμα στον παραλήπτη και ο τελευταίος μπορεί να το αποκρυπτογραφήσει με το ιδιωτικό κλειδί του. Δεδομένου ότι το ιδιωτικό κλειδί του παραλήπτη είναι γνωστό μονάχα στον ίδιο και σε κανέναν άλλον, μονάχα ο παραλήπτης μπορεί να αποκρυπτογραφήσει το μήνυμα και να το διαβάσει. Άρα λοιπόν, με αυτόν τον τρόπο ο αποστολέας και ο παραλήπτης γνωρίζει ότι το κρυπτογραφημένο μήνυμα μπορεί να αποκρυπτογραφηθεί μονάχα από τον παραλήπτη και έτσι, διασφαλίζεται η εμπιστευτικότητα του μηνύματος.

Προκειμένου να επιτευχθεί η πιστοποίηση θα πρέπει ο αποστολέας να χρησιμοποιήσει το ιδιωτικό του κλειδί για την κρυπτογράφηση του μηνύματος. Στη συνέχεια στέλνει το μήνυμα στον παραλήπτη και ο τελευταίος χρησιμοποιεί το δημόσιο κλειδί του αποστολέα για την κρυπτογράφησή του. Παίρνοντας σαν δεδομένο ότι το ιδιωτικό κλειδί του αποστολέα είναι γνωστό μονάχα στον ίδιο, ο παραλήπτης μπορεί να είναι σίγουρος για την ταυτότητα του αποστολέα.

Το μέγεθος του κλειδιού αποτελεί τον κύριο παράγοντα εξασφάλισης της μυστικότητας του περιεχομένου του κρυπτογραφημένου μηνύματος, δεδομένου ότι ο αλγόριθμος που χρησιμοποιείται είναι ασφαλής. Ο μόνος τρόπος αποκρυπτογράφησης ενός μηνύματος, το οποίο έχει κρυπτογραφηθεί με έναν ασφαλή αλγόριθμο, είναι να δοκιμαστούν όλοι οι δυνατοί συνδυασμοί του κλειδιού. Κατά συνέπεια, αν το κλειδί είναι μεγάλο σε μέγεθος κάτι τέτοιο καθίσταται αδύνατο.

Αν προσπαθήσουμε να εισάγουμε και μια νέα έννοια, την χρηματική αξία του κλειδιού θα καταλάβουμε ότι η διαδικασία της αποκρυπτογράφησης δεν είναι τόσο απλή. Προκειμένου ένας τρίτος να αφιερώσει χρόνο και υπολογιστικούς πόρους για να κάνει κρυπτανάλυση σε ένα μήνυμα, θα πρέπει το περιεχόμενο του μηνύματος να έχει και αντίστοιχη χρηματική αξία. Αν θεωρήσουμε ότι ένας αλγόριθμος που χρησιμοποιείται για την κρυπτογράφηση ενός μηνύματος είναι ασφαλής, δηλαδή δεν μπορεί να παραβιαστεί και ο μόνος τρόπος αποκρυπτογράφησης είναι η χρήση του σωστού κλειδιού, τότε κάποιος που θέλει να αποκτήσει πρόσβαση στο περιεχόμενο είναι αναγκασμένος να δοκιμάσει όλους τους δυνατούς

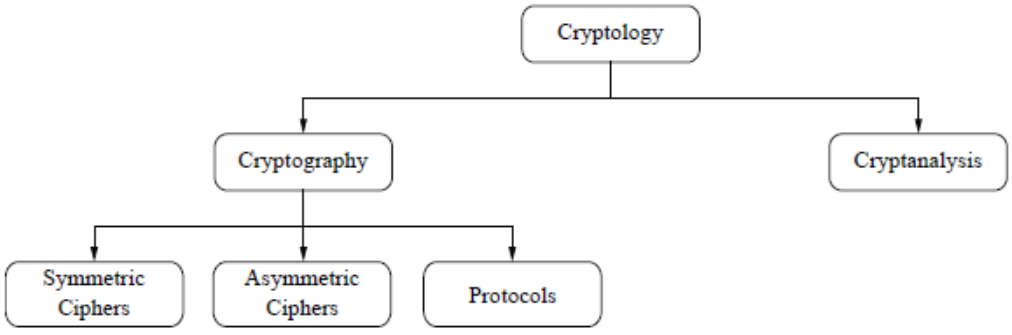
συνδυασμούς έως ότου βρεθεί το σωστό κλειδί. Ενδεικτικά στον παρακάτω πίνακα αναγράφονται οι απαιτούμενοι χρόνοι καθώς και το κόστος για να γίνει αυτής της μορφής η κρυπτανάλυση. [16]

Προσεγγιστικός χρόνος κρυπτανάλυσης					
Κόστος	40 bit	56 bit	64 bit	80 bit	128 bit
100.000 €	2 sec	35hrs	1yr	70.000 yrs	10^{19} yrs
1.000.000 €	0.2 sec	3.5hrs	37 days	7.000 yrs	10^{18}yrs
10.000.000 €	0.02 sec	1260 sec	9 hrs	700 yrs	10^{17}yrs
100.000.000 €	0.002 sec	120 sec	4 hrs	70 yrs	10^{16}yrs

Πίνακας 2.1 Προσεγγιστικοί χρόνοι κρυπτανάλυσης

2.6 Κρυπτανάλυση

Η κρυπτανάλυση είναι η μελέτη για την επινόηση μεθόδων που εξασφαλίζουν την κατανόηση του νοήματος της κρυπτογραφημένης πληροφορίας, έχοντας ως άγνωστη παράμετρο το κλειδί με βάση το οποίο πραγματοποιήθηκε η κρυπτογράφηση .Βασικός στόχος της είναι η εύρεση του κλειδιού ή ενός ισοδύναμου αλγόριθμου που θα βοηθήσει ώστε να αναγνωστεί η κρυπτογραφημένη πληροφορία. Ένας κρυπτογραφικός αλγόριθμος θεωρείται ότι έχει “σπάσει” όταν βρεθεί μια μέθοδος (πιθανολογική ή ντετερμινιστική) που θα μπορέσει να βρει το μήνυμα ή το κλειδί με χρονική πολυπλοκότητα μικρότερη από την πολυπλοκότητα της επίθεσης ωμής βίας. Στην πραγματικότητα, όπως φαίνεται και από την εικόνα 3.1, η κρυπτανάλυση και η κρυπτογραφία είναι δύο έννοιες που συνδέονται στην γενικότερη επιστήμη της κρυπτολογίας.



Εικόνα *4.1* Γενική επισκόπηση της κρυπτολογίας

Ο λόγος που οι δύο αυτοί κλάδοι συνδέονται είναι το γεγονός ότι δεν μπορούμε να βεβαιωθούμε για την ασφάλεια ενός κρυπτογραφικού αλγόριθμου αν δεν εφαρμόσουμε μεθόδους κρυπτανάλυσης.

2.7 Είδη επιθέσεων κρυπτανάλυσης

1. Επίθεση ωμής βίας (brute force attack)

Είναι η επίθεση η οποία δοκιμάζει όλα τα πιθανά κλειδιά που παράγουν ένα κρυπτογράφημα έτσι ώστε να αποκαλυφθεί το αρχικό μήνυμα. Τέτοιου είδους επιθέσεις μπορούν να πραγματοποιηθούν σε οποιονδήποτε αλγόριθμο κρυπτογράφησης. Συνήθως ο επιτιθέμενος χρησιμοποιεί αρχικά κάποια πιθανά κατά αυτόν κλειδιά έτσι ώστε να ανακτήσει πιο γρήγορα το κλειδί. Η μέθοδος αυτή λειτουργεί περισσότερο σαν μέτρο σύγκρισης για τις υπόλοιπες επιθέσεις (όσον αφορά τον χρόνο που χρειάζεται η καθεμιά). Είναι προφανές ότι όσο μεγαλύτερο είναι το κλειδί που χρησιμοποιείται, τόσο πιο δύσκολη καθίσταται η συγκεκριμένη επίθεση. Π.χ. ένας κρυπτογραφικός αλγόριθμος με μήκος κλειδιού 3 bits έχει $2^3 = 8$ διαφορετικούς συνδυασμούς ενώ αν χρησιμοποιήσουμε ένα κλειδί με μήκος 64 bits θα έχουμε $2^{64} = 18{,}446{,}744{,}073{,}709{,}551{,}616$ διαφορετικούς συνδυασμούς.

2. Ciphertext-only attack

Στην περίπτωση αυτής της επίθεσης, ο κρυπταναλυτής έχει το κρυπτοκείμενο από πολλά μηνύματα τα οποία έχουν κρυπτογραφηθεί με τον ίδιο αλγόριθμο κρυπτογράφησης. Ο σκοπός του είναι να ανακτήσει όσο το δυνατόν περισσότερα αρχικά μηνύματα ή να καταφέρει να ανακτήσει το κλειδί που χρησιμοποιήθηκε κατά την διαδικασία της κρυπτογράφησης.

3. Known-Plaintext attack

Στην περίπτωση αυτή ο κρυπταναλυτής έχει πρόσβαση όχι μόνο σε κάποια από τα κρυπτογραφημένα μηνύματα μιας συνομιλίας αλλά και στα αρχικά μηνύματα από τα οποία προήλθαν. Η δουλειά του είναι, με βάση αυτά τα δεδομένα, να συμπεράνει την τιμή του κλειδιού με το οποίο έγινε η κρυπτογράφηση ή να μπορέσει να φτιάξει έναν αλγόριθμο ο οποίος θα μπορεί να αποκρυπτογραφήσει οποιαδήποτε μελλοντικά μηνύματα της συγκεκριμένης συνομιλίας.

4. Chosen Plaintext attack

Ο κρυπταναλυτής έχει πρόσβαση στο αρχικό κείμενο, στο κρυπτογραφημένο κείμενο αλλά μπορεί επίσης και να επιλέξει το κείμενο το οποίο θα κρυπτογραφηθεί. Αυτή είναι μια πιο ισχυρή επίθεση καθώς ο κρυπταναλυτής μπορεί να επιλέξει συγκεκριμένα μηνύματα για να κρυπτογραφήσει, αυτά τα οποία θα μπορούν να του δώσουν περισσότερες πληροφορίες σχετικά με το κλειδί.

Πιο σύγχρονες μέθοδοι για κρυπτανάλυση συστημάτων είναι η διαφορική κρυπτανάλυση (differential cryptanalysis), γραμμική κρυπτανάλυση (linear cryptanalysis), στατιστική κρυπτανάλυση (statistical cryptanalysis) και η μέθοδος των πινάκων ουράνιου τόξου (rainbow tables) την οποία θα αναλύσουμε εκτενέστερα σε αυτή την εργασία καθώς και την εφαρμογή της στον αλγόριθμο A5/3.

2.8 Ορολογία

Στο υπόλοιπο κείμενο θα χρησιμοποιήσουμε κάποιες λέξεις που απαντώνται στην κρυπτογραφία οπότε θα ήταν καλό να τις διευκρινίσουμε από τώρα.

- Κρυπτογράφηση (encryption) : είναι η διαδικασία κατά την οποία μετατρέπουμε ένα κείμενο (plaintext) σε μια μη κατανοητή μορφή (cipher text) έτσι ώστε να μην είναι δυνατή η ανάγνωση του κειμένου από κάποιον εκτός του παραλήπτη.
- Κρυπτογραφικός αλγόριθμος (cryptographic algorithm ή cipher) : είναι ο αλγόριθμος που χρησιμοποιείται για την παραπάνω μετατροπή. Ο αλγόριθμος αυτός θα πρέπει να υπακούει σε κάποιους κανόνες τους οποίους αναλύουμε παρακάτω.
- Κλειδί (key) : Το κλειδί είναι η βασική παράμετρος για έναν κρυπταλγόριθμο, η οποία καθορίζει την έξοδό του. Η έννοια του κλειδιού χρησιμοποιείται επίσης και σε άλλες εφαρμογές όπως ψηφιακές υπογραφές, κωδικοί πιστοποίησης μηνύματος κλπ.
- Κρυπτανάλυση (cryptanalysis) : Είναι η μελέτη των μεθόδων για την κατανόηση της κρυπτογραφημένης πληροφορίας χωρίς την χρήση του μυστικού κλειδιού. Τεχνικές που χρησιμοποιούνται στην κρυπτανάλυση περιγράφονται παρακάτω.

3 Ο ΚΡΥΠΤΟΓΡΑΦΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ A5/3

3.1 Κρυπτογραφία και κρυπτανάλυση στην κινητή τηλεφωνία.

Υπάρχουν πολλοί αλγόριθμοι κρυπτογράφησης που χρησιμοποιούνται κυρίως στον τομέα των ασύρματων επικοινωνιών και πολλοί οι οποίοι αναπτύσσονται. Ένας τρόπος για να προστατευτεί ένας αλγόριθμος κρυπτογράφησης είναι η μη δημοσιοποίηση του τρόπου λειτουργίας του παρά μόνον σε συγκεκριμένους κατασκευαστές, οι οποίοι θα πρέπει να τον υλοποιήσουν. Η συγκεκριμένη τακτική έχει αποδειχτεί δύσκολη στην πράξη καθώς με την τεχνική του reverse engineering πολλοί τέτοιοι αλγόριθμοι έχουν δημοσιευτεί.

Ωστόσο, ένας καλύτερος τρόπος προστασίας είναι η δημοσιοποίηση του αλγορίθμου έτσι ώστε η ερευνητική κοινότητα της κρυπτανάλυσης να ασχοληθεί με τον έλεγχό του για ορθή λειτουργία. Αν ένας αλγόριθμος μπορεί να περάσει το συγκεκριμένο τεστ τότε είναι αρκετά πιο σίγουρη η ασφάλεια που παρέχει.

Στην κινητή τηλεφωνία και συγκεκριμένα στα δίκτυα GSM (Global System for Mobile communications) και GPRS (General Packet Radio Service), η ασφάλεια που παρέχεται στον χρήστη περικλείεται στην κάρτα SIM του και στην ίδια την συσκευή. Η κάρτα SIM περιέχει ένα κλειδί πιστοποίησης (μήκους 128 bits) με το οποίο ελέγχεται αν ο χρήστης έχει την άδεια να χρησιμοποιήσει το δίκτυο. Το κλειδί πιστοποίησης που περιέχει η κάρτα SIM κρυπτογραφείται μέσω του αλγορίθμου A8 (ο οποίος είναι υλοποιημένος πάνω στην κάρτα) έτσι ώστε να δώσει σαν έξοδο ένα κλειδί (μήκους 64 bits) που θα χρησιμοποιηθεί για την κωδικοποίηση δεδομένων κατά την διάρκεια της μετάδοσης πληροφορίας από το κινητό (γραπτό μήνυμα, φωνή κλπ.).

Τα δίκτυα κινητής τηλεφωνίας παρέχουν 2 στάδια ασφάλειας : τον αλγόριθμο A5/2 και τον A5/1 (σε σειρά αύξουσας παροχής ασφάλειας). Οι αλγόριθμοι αυτοί δεν είχαν δημοσιευτεί και μετά από χρόνια, με την τεχνική του reverse engineering, έγιναν γνωστοί. Για να λειτουργήσουν οι αλγόριθμοι αυτοί, χρησιμοποιούν το κλειδί που προαναφέραμε (64-bits) που προέρχεται από την κάρτα SIM. Για τον A5/1 έχουν δημοσιευτεί πολλές επιθέσεις, με κυριότερη αυτή των Chris Paget και Karsten Nohl, οι οποίοι δημοσίευσαν τα rainbow tables που μπορούσαν να “σπάσουν” τον A5/1. Σχετικά με τον A5/2, έχει αποδειχτεί πολύ λιγότερο ασφαλής και η εύρεση του μυστικού κλειδιού για αυτόν γίνεται σε λιγότερο από ένα δευτερόλεπτο χρησιμοποιώντας έναν κοινό υπολογιστή.

Οι αλγόριθμοι της σειράς A5 χρησιμοποιούνται για να κρυπτογραφήσουν αλλά και να αποκρυπτογραφήσουν τα δεδομένα τα οποία μεταδίδονται μεταξύ των κινητών. Επίσης, οι σταθμοί βάσης (Base Transceiver Stations, BTS) χρησιμοποιούν τους αλγόριθμους της σειράς A5. Συνοπτικά για τους αλγόριθμους A5 έχουμε:

- A5/1 : Είναι μέχρι σήμερα ο standard κρυπτογραφικός αλγόριθμος για τα δίκτυα της Ευρώπης και των Η.Π.Α. Είναι ένα stream cipher και έχουν πραγματοποιηθεί κατά καιρούς πολλές επιθέσεις εναντίον του λόγω της μεγάλης του εφαρμογής.

- A5/2 : Η σκοπίμως αποδυναμωμένη έκδοση του αλγορίθμου A5/1 που σχεδιάζεται να χρησιμοποιηθεί σε μη δυτικές χώρες. Ο A5/2 είναι stream cipher.
- A5/3 : Δεν έχει γίνει ακόμα standard, χρησιμοποιείται για μετάδοση 3g δεδομένων, είναι ένα block cipher και εξετάζεται η ασφάλειά του.

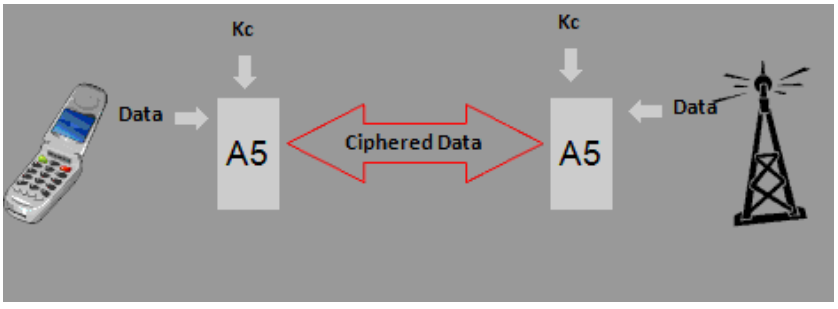
Λόγω των παραπάνω κενών ασφαλείας που παρατηρήθηκαν λοιπόν, αναπτύχθηκε ο αλγόριθμος A5/3 ο οποίος δεν έχει γίνει ακόμα standard λόγω του γεγονότος ότι εμφανίζει αδυναμίες στην ασφάλεια που παρέχει.

3.2 Γενικά για τον κρυπταλγόριθμο

Ο αλγόριθμος A5/3 υιοθετήθηκε το 2000 από την 3GPP (3rd generation partnership project), και είχε ως στόχο την υλοποίηση των αλγορίθμων εμπιστευτικότητας (confidentiality) και ακεραιότητας (integrity) δεδομένων για τα δίκτυα κινητής τηλεφωνίας τρίτης γενιάς (3g).

Λόγω της αναγκαιότητας ύπαρξης ενός standard για την κρυπτογραφία σε δίκτυα τρίτης γενιάς, ο αλγόριθμος A5/3, αντί να σχεδιαστεί από την αρχή , στηρίχτηκε σε έναν ήδη υπάρχον αλγόριθμο κρυπτανάλυσης. Ο βασικός λόγος ήταν ότι έπρεπε σχετικά γρήγορα να υιοθετηθεί ένας standard αλγόριθμος για κρυπτογράφηση και η χρήση ενός παλιότερου, ο οποίος είχε περάσει ήδη από πολλές αξιολογήσεις, φαινόταν ιδανική.

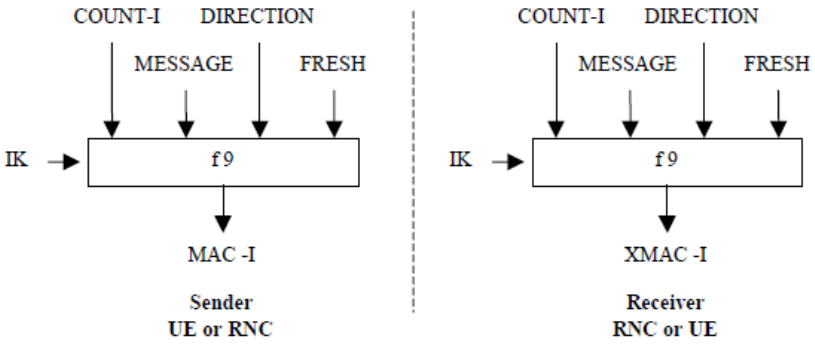
Κατά συνέπεια, σαν βάση χρησιμοποιήθηκε ο αλγόριθμος MISTY1 της Mitsubishi Electric Corporation, ο οποίος είχε δημιουργηθεί το 1995. Οι αλλαγές που έγιναν ήταν ελάχιστες, και αφορούσαν κυρίως την ευκολότερη υλοποίηση σε hardware, καθώς και την συμβατότητα με τα standard που υπήρχαν ήδη για τα δίκτυα τρίτης γενιάς. Από τον αλγόριθμο MISTY1 πήρε και την κοινή του ονομασία ο A5/3, είναι κυρίως γνωστός με το όνομα Kasumi (στα ιαπωνικά σημαίνει mist=θολό).



Εικόνα 3.1 Γενική αρχιτεκτονική του τρόπου μεταφοράς κρυπτογραφημένων δεδομένων

Ο αλγόριθμος A5/3 χρησιμοποιείται στο πρωτόκολλο UMTS R99 για δύο κύριους λόγους: για την εμπιστευτικότητα των δεδομένων που στέλνονται (data confidentiality) αλλά και για την ακεραιότητα των δεδομένων αυτών (integrity protection). Την εμπιστευτικότητα την υλοποιεί η συνάρτηση f8 και την ακεραιότητα η συνάρτηση f9 (σχήμα 2.2). Η συνάρτηση f9 για

παράδειγμα αναλαμβάνει να δημιουργήσει ένα message authentication code (MAC) το οποίο ελέγχεται για την ορθότητά του στον receiver.

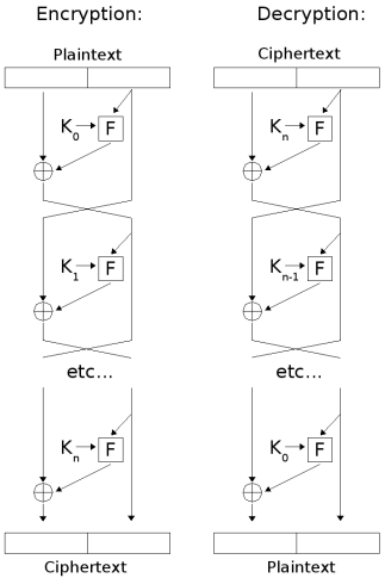


Εικόνα 3.2 Δημιουργία του MAC για ένα μήνυμα

Οι υλοποιήσεις των συναρτήσεων f8 και f9 βασίζονται στον κρυπτογραφικό αλγόριθμο A5/3. Για τον λόγο αυτό, στην εργασία αυτή δεν θα ασχοληθούμε με τις ίδιες τις συναρτήσεις αλλά με τον κρυπτογραφικό αλγόριθμο καθώς αυτός είναι που μας ενδιαφέρει κυρίως.

3.3 Τεχνικές προδιαγραφές

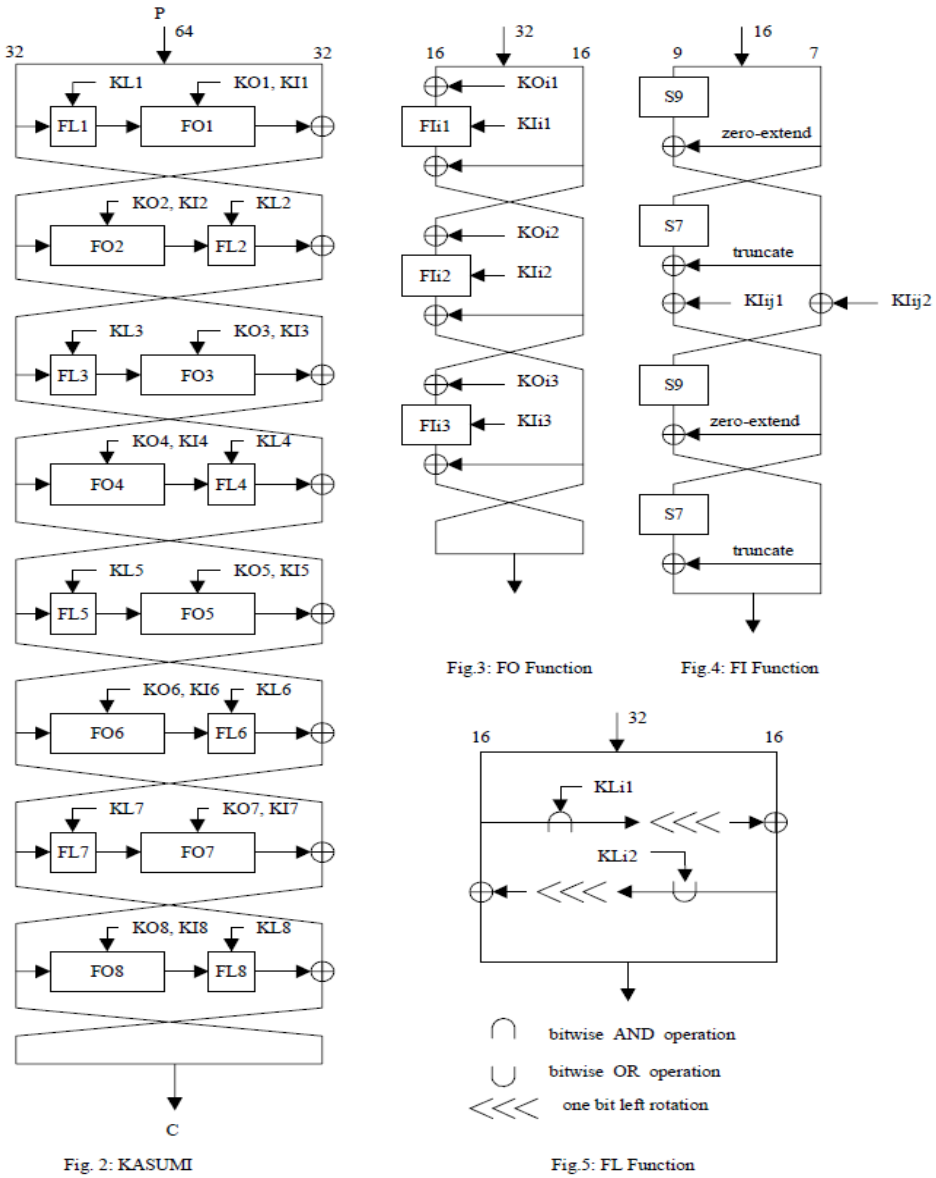
Ο A5/3 είναι αλγόριθμος συμμετρικού κλειδιού, μήκους 128-bits, και η είσοδος όσο και η έξοδός του είναι 64-bits. Η αρχιτεκτονική που ακολουθεί είναι δομής Feistel, 8 γύρων. Η δομή Feistel μας εξασφαλίζει εύκολη διαδικασία αποκρυπτογράφησης όπως φαίνεται και από την παρακάτω εικόνα.



Εικόνα 3.3 Κλασική δομή ενός αλγορίθμου αρχιτεκτονικής Feistel

Στην διαδικασία της αποκρυπτογράφησης συνήθως το μόνο που χρειάζεται να αλλάξουμε είναι τα κλειδιά που χρησιμοποιούμε και να τα εισάγουμε με αντίστροφη σειρά. Αν εφαρμόσουμε αυτή την διαδικασία τότε, μπορούμε να αναπαράγουμε το αρχικό, εισαγόμενο κείμενο (plaintext).

Όπως μπορούμε να διαπιστώσουμε και από το σχήμα 2.4, το εισαγόμενο κείμενο (plaintext, P) είναι μήκους 64 bits, όπως και το κρυπτογραφημένο κείμενο (ciphertext, C). Το εισαγόμενο κείμενο περνάει 8 στάδια τα οποία είναι ανά δύο όμοια μεταξύ τους. Μπορούμε δηλαδή να κατηγοριοποιήσουμε τους 8 γύρους του A5/3 σε μονούς και ζυγούς γύρους. Στους μονούς γύρους (1,3,5,7) η συνάρτηση FO προηγείται της FL, ενώ στους ζυγούς γύρους (0,2,4,6) συμβαίνει το αντίθετο.



Εικόνα 3.4 **Kasumi block diagram**

Όταν ξεκινάει ο αλγόριθμος, το εισαγόμενο κείμενο χωρίζεται σε δύο 32-bits ποσότητες L και R, έτσι ώστε το concatenation των L και R να μας δίνει το P. Έπειτα, για τους γύρους 1<= I <=8 ισχύει :

$$R_i = L_{i-1}$$

$$L_i = R_{i-1} \oplus f_i \; L_{i-1}, RK_i$$

με τα L_i και R_i να αποτελούν την είσοδο για τον εκάστοτε γύρο i. Το RK_i αποτελεί το round key του κάθε γύρου και θα αναφερθούμε αργότερα στο πως παράγεται, δηλαδή στο key scheduling του αλγορίθμου μας.

Συνάρτηση FL

Η συνάρτηση FL είναι μια γραμμική συνάρτηση και η ασφάλεια του αλγορίθμου δεν εξαρτάται από αυτήν. Η είσοδος και η έξοδός της είναι μήκους 32 bits. Ο βασικός της σκοπός είναι να αναμίξει περισσότερο τα bits έτσι ώστε να είναι ακόμα πιο δύσκολο να τα παρακολουθήσει κάποιος κατά την διάρκεια των 8 γύρων.

Η είσοδος της συγκεκριμένης συνάρτησης χωρίζεται σε δύο 16-bits ποσότητες L και R και αν υποθέσουμε ότι η έξοδος είναι της μορφής $R' \parallel L'$ τότε, η έξοδος δίνεται από τις σχέσεις

$$R' = R \oplus ROL(L \cap KI_{i,1})$$

$$L' = L \oplus ROL(R' \cup KI_{i,2})$$

όπου το σύμβολο ROL εκφράζει την αριστερή κυκλική ολίσθηση κατά 1 bit.

Η συνάρτηση FL έχει την βασική ιδιότητα ότι για μια είσοδο της μορφής $0^{16}1^{16}$, και για οποιαδήποτε τιμή κλειδιού, δίνει σαν αποτέλεσμα 1^{32} . Συνεπώς, για κάποιες εισόδους (της μορφής $0^{16}1^{16}$), μερικά bits του κλειδιού μπορούν να αλλαχτούν χωρίς αυτό να έχει επίπτωση στο αποτέλεσμα του γύρου.

Συνάρτηση FI

Η συνάρτηση αυτή είναι η βασική συνάρτηση του A5/3, η είσοδος και η έξοδός της είναι μεγέθους 16-bits. Αποτελείται από μια δομή τεσσάρων γύρων και περιέχει δύο μη γραμμικά μπλοκ αντικατάστασης (substitution boxes), τα S7 και S9.

Τα μπλοκ αντικατάστασης έχουν σχεδιαστεί έτσι ώστε να μπορούν να υλοποιηθούν εύκολα σε hardware τόσο με συνδυαστική λογική όσο και με ένα look-up table. Αυτό που κάνουν τα μπλοκ S7 και S9 είναι να μετατρέπουν μια ακολουθία x (μήκους 7 ή 9 bits αντίστοιχα)

σε μια ακολουθία y. Αν η ακολουθία x είναι της μορφής

$$x = x8 \parallel x7 \parallel x6 \parallel x5 \parallel x4 \parallel x3 \parallel x2 \parallel x1 \parallel x0$$

όπου τα $x8 \ldots x0$ αναπαριστούν τις τιμές των bits (προφανώς τα $x8$ και $x7$ χρησιμοποιούνται μόνο στο S9 box). Τα αντίστοιχα bits της εξόδου, $y8 \ldots y0$, παράγονται με έναν συνδυασμό λογικών πράξεων XOR και AND μεταξύ των bits $x8 \ldots x0$. Λεπτομέρειες για τον ακριβή τρόπο υπολογισμού τους, βρίσκονται στο [2].

Αν η είσοδος της συνάρτησης FI είναι της μορφής $L_0 \parallel R_0$ (το L_0 είναι 9 bits ενώ το R_0 7 bits) τότε η έξοδος θα δίνεται από τις εξισώσεις

$$L_1 = R_0 \quad R_1 = S9\ L_0 \ \oplus ZE(R_0)$$

$$L_2 = R_1 \oplus KI_{i,j,2} \quad R_2 = S_7\ (L_1) \oplus TR(\ R_1) \oplus KI_{i,j,1}$$

$$L_3 = R_2 \quad R_3 = S9[L_2] \oplus ZE(R_2)$$

$$L_4 = S7\ L_3 \ \oplus TR(R_3) \qquad R_4 = R_3$$

Η έξοδος θα είναι η $L_4 \parallel R_4$.

Δημιουργία Κλειδιού

Η δημιουργία κλειδιού του αλγορίθμου A5/3 είναι πολύ απλή χωρίς αυτό να αποτελεί αδυναμία, ούτε ότι με το να γινόταν πιο πολύπλοκη θα υπήρχε κάποιο κέρδος στην ασφάλεια του αλγορίθμου. Το κάθε ένα από τα 128 bits του κρυφού κλειδιού χρησιμοποιείται μία και μόνο φορά στον κάθε γύρο (από τους 8 συνολικά). Κάθε bit χρησιμοποιείται με διαφορετικό τρόπο, σε διαφορετικούς γύρους και μάλιστα και σε διαφορετικά μέρη του κάθε γύρου αφού πρώτα μεταβληθεί ύστερα από λογική πράξη με κάποια σταθερά.

Πιο αναλυτικά, η παραγωγή του κλειδιού γίνεται με τον εξής τρόπο : Έστω K το αρχικό κλειδί μας (128 bits), το οποίο το χωρίζουμε σε 8 16-bits ποσότητες ως

$$K = K1 \parallel K2 \parallel K3 \parallel \cdots \parallel K8$$

Δημιουργούμε έναν πίνακα με 8 υπο-κλειδιά, K_i' τα οποία δημιουργούνται από τα K1 … K8 με τον τύπο

$$K_i' = C_i \oplus K_i \quad , (1 \leq i \leq 8)$$

όπου η σταθερά C_i είναι συγκεκριμένη δυαδική τιμή μήκους 16-bits. Ο υπολογισμός του υπο-κλειδιού που χρησιμοποιείται σε κάθε γύρο γίνεται με τον τρόπο που φαίνεται στον παρακάτω πίνακα.

Round	$KL_{i,1}$	$KL_{i,2}$	$KO_{i,1}$	$KO_{i,2}$	$KO_{i,3}$	$KI_{i,1}$	$KI_{i,2}$	$KI_{i,3}$
1	$K_1 \lll 1$	K'_3	$K_2 \lll 5$	$K_6 \lll 8$	$K_7 \lll 13$	K'_5	K'_4	K'_8
2	$K_2 \lll 1$	K'_4	$K_3 \lll 5$	$K_7 \lll 8$	$K_8 \lll 13$	K'_6	K'_5	K'_1
3	$K_3 \lll 1$	K'_5	$K_4 \lll 5$	$K_8 \lll 8$	$K_1 \lll 13$	K'_7	K'_6	K'_2
4	$K_4 \lll 1$	K'_6	$K_5 \lll 5$	$K_1 \lll 8$	$K_2 \lll 13$	K'_8	K'_7	K'_3
5	$K_5 \lll 1$	K'_7	$K_6 \lll 5$	$K_2 \lll 8$	$K_3 \lll 13$	K'_1	K'_8	K'_4
6	$K_6 \lll 1$	K'_8	$K_7 \lll 5$	$K_3 \lll 8$	$K_4 \lll 13$	K'_2	K'_1	K'_5
7	$K_7 \lll 1$	K'_1	$K_8 \lll 5$	$K_4 \lll 8$	$K_5 \lll 13$	K'_3	K'_2	K'_6
8	$K_8 \lll 1$	K'_2	$K_1 \lll 5$	$K_5 \lll 8$	$K_6 \lll 13$	K'_4	K'_3	K'_7

$(X \lll i)$ — X rotated to the left by i bits

Εικόνα 3.5 Kasumi key scheduling

Λόγω της χρησιμοποίησης των σταθερών C1 … C8 δεν υπάρχει σχέση επανάληψης μεταξύ συνεχόμενων υπο-κλειδιών. Αυτή η ιδιότητα είναι απαραίτητη καθώς αποτρέπει συγκεκριμένες επιθέσεις που μπορούν να γίνουν στον κρυπταλγόριθμο. Οι σταθερές αυτές καθώς και περισσότερες λεπτομέρειες πάνω στον αλγόριθμο μπορούν να βρεθούν στο [2].

3.4 Βελτίωση του αλγορίθμου: μετατροπή σε pipelined σχεδίαση

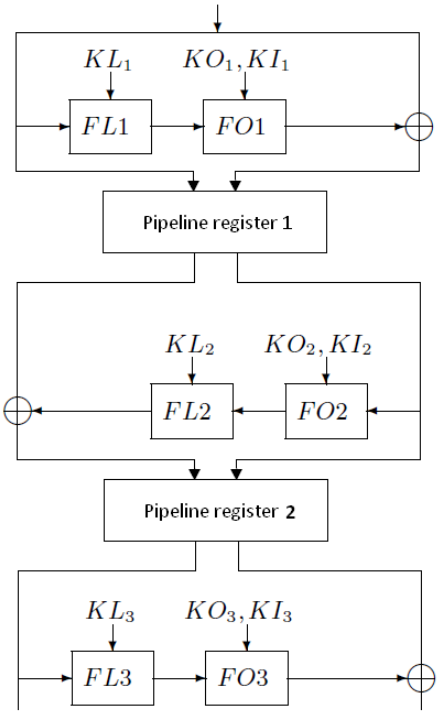
Προκειμένου να βελτιστοποιήσουμε την ταχύτητα του αλγορίθμου μας, υλοποιήσαμε μια pipelined μορφή του. Με την σχεδίαση αυτή, θα μπορούμε σε κάθε χρονική στιγμή να εισάγουμε και ένα καινούριο plaintext στον αλγόριθμο (ή και ένα καινούριο κλειδί). Πιο συγκεκριμένα:

Μετά από κάθε γύρο του αλγορίθμου μας, δημιουργήσαμε έναν pipelined καταχωρητή ο οποίος παίρνει την έξοδο του προηγούμενου γύρου και την προωθεί στον επόμενο. Με τον τρόπο αυτό ανεξαρτητοποιούνται ουσιαστικά οι γύροι μεταξύ τους.

Αν για παράδειγμα μία χρονική στιγμή ενεργοποιήσουμε τον αλγόριθμό μας, αυτός θα πάρει το plaintext και το key που θα δει στην είσοδό του και θα υπολογίσει το αποτέλεσμα του $1^{ου}$ γύρου του αλγορίθμου μας. Το αποτέλεσμα αυτό εγγράφεται στον $1^{ο}$ pipeline καταχωρητή μας. Την επόμενη χρονική στιγμή (στον επόμενο χτύπο ρολογιού δηλαδή), μεταφέρουμε τα δεδομένα του πρώτου καταχωρητή στον δεύτερο γύρο του αλγορίθμου μας. Ταυτόχρονα όμως, η σχεδίασή μας είναι έτοιμη να δεχτεί και άλλη είσοδο (είτε plaintext είτε κλειδί) και να υπολογίσει καινούριο αποτέλεσμα $1^{ου}$ γύρου το οποίο θα γραφτεί στον πρώτο καταχωρητή. Η διαδικασία αυτή μας προσφέρει μια επιτάχυνση της τάξης του x8 αν υπάρχει συνεχής ροή δεδομένων στην είσοδο του αλγορίθμου.

Ο κύριος λόγος που μπορούμε να υλοποιήσουμε μια pipelined αρχιτεκτονική για την σχεδίασή μας είναι η ανεξαρτησία των δεδομένων μεταξύ των 8 γύρων του αλγορίθμου μας. Τα δεδομένα που προκύπτουν για παράδειγμα από τον πρώτο γύρο δεν μας χρειάζονται άμεσα για να υπολογίσουμε τα δεδομένα του $4^{ου}$ γύρου. Στην πραγματικότητα, κάθε γύρος χρειάζεται

μόνο τα δεδομένα που προκύπτουν από τον προηγούμενο από αυτόν γύρο. Κατά συνέπεια, μπορούν να υπάρχουν ταυτόχρονα στην σχεδίασή μας 8 διαφορετικά plaintexts χωρίς να επηρεάζει το ένα το άλλο.



Εικόνα 3.6 Pipelined αρχιτεκτονική του αλγορίθμου μας

3.5 Γνωστές επιθέσεις στον κρυπτογραφικό αλγόριθμο

Το 2003 οι ερευνητές Elad Barkan, Eli Biham και Nathan Keller παρουσίασαν επιθέσεις στο πρωτόκολλο GSM οι οποίες επέτρεπαν την αποφυγή του κρυπταλγόριθμου A5/3 και κατά συνέπεια, ήταν εφικτό το σπάσιμο του πρωτοκόλλου GSM. Η επίθεση αυτή ωστόσο δεν είχε να κάνει άμεσα με τον κρυπταλγόριθμο A5/3.

Το 2005 οι ερευνητές Eli Biham, Orr Dunkelman και Nathan Keller δημοσίευσαν μια επίθεση συσχετιζόμενου κλειδιού για τον A5/3 η οποία μπορούσε να σπάσει και τους 8 γύρους του κρυπταλγόριθμου πιο γρήγορα από μια εξονυχιστική αναζήτηση. Η επίθεση αυτή απαιτούσε $2^{54.6}$ επιλεγμένες ακρυπτογράφητες λέξεις. Στην συγκεκριμένη επίθεση, η καθεμία από αυτές κρυπτογραφείται με βάση ένα από τα 4 συσχετιζόμενα κλειδιά που χρησιμοποιούνται και έχει χρονική πολυπλοκότητα της τάξης του $2^{76,1}$ κρυπτογραφήσεις του αλγορίθμου. Αν και η συγκεκριμένη επίθεση δεν είναι τόσο πρακτική, λόγω μεγάλης χρονικής πολυπλοκότητας, αναδεικνύει κάποιες αδυναμίες του αλγορίθμου A5/3.

Το 2010, οι ερευνητές Orr Dunkleman, Nathan Keller και Adi Shamir δημοσίευσαν μια νέα επίθεση συσχετιζόμενου κλειδιού η οποία μπορούσε να ανακτήσει ολόκληρο το κλειδί του κρυπταλγόριθμου.

3.6 Αδυναμίες του αλγορίθμου A5/3

Μια βασική αδυναμία του αλγορίθμου είναι ότι αν και το μήκος κλειδιού του είναι 128 bits, το αποτελεσματικό του μήκος (effective key length) είναι 64 bits. Ο λόγος είναι ότι για να δημιουργηθεί αυτό το κλειδί, απλά επαναλαμβάνεται το κλειδί που προέρχεται από την κάρτα SIM (μήκους 64 bits). Η ανάγκη για back-end συμβατότητα επιβάλλει αυτή την επανάληψη μιας και οι συσκευές κινητής τηλεφωνίας σχεδιάζονται έτσι ώστε να μπορούν να λειτουργήσουν σε οποιοδήποτε δίκτυο και αν βρεθούν.

Στην πραγματικότητα αυτή η συμβατότητα δημιουργεί μεγάλα προβλήματα ασφαλείας καθώς είναι εύκολο να παρακαμφτεί η ασφάλεια του A5/3. Μια συσκευή μπορεί να εξαναγκαστεί να πιστέψει ότι το δίκτυο στο οποίο είναι συνδεδεμένη υποστηρίζει μόνο τον αλγόριθμο κρυπτογράφησης A5/2 και κατά συνέπεια η εύρεση του μυστικού κλειδιού καθίσταται πιο εύκολη.

3.7 Hardware υλοποίηση του Αλγορίθμου A5/3

Για να υλοποιήσουμε σε hardware τον αλγόριθμο A5/3, χρησιμοποιήσαμε τα specifications από την 3rd Generation Partner Project. Τα modules άλλωστε που περιγράφονται στα specifications είναι φτιαγμένα ώστε να υλοποιούνται καλύτερα σε hardware.

Τα αποτελέσματα του synthesize του top level του αλγορίθμου μας είναι τα εξής (για μια Virtex5 XC5VLX330T)

Minimum period: 9.099 nsec (Maximum frequency 109.900 Mhz)

Number of Slice LUTs : 3,866/207,360 Utilization : 1%

Συνολικά λοιπόν, για να τρέξει μία φορά ο αλγόριθμός μας σε hardware έτσι ώστε να παραχθεί ένα cipher text, χρειάζονται

$$\text{Clock cycle: } 9.099 \text{ nsec} \times 8 \text{ cycles} = 72.792 \text{ nsec}$$

Η ορθότητα της σχεδίασής μας ελέγχθηκε από τα Implementer' test data της 3GPP [4].

3.8 Software υλοποίηση του Αλγορίθμου A5/3

Για να υλοποιήσουμε σε software τον αλγόριθμο A5/3 χρησιμοποιήσαμε το [14] που παρέχεται από την 3GPP. Το συγκεκριμένο report μας έδινε τις συναρτήσεις που χρειάζονται για τον αλγόριθμο (FL,FI,FO, key scheduling) και εμείς φτιάξαμε ένα πρόγραμμα στην γλώσσα C που να χρησιμοποιεί τις συναρτήσεις αυτές μέσα από μια main function.

Προκειμένου να δούμε την απόδοση που έχει ο συγκεκριμένος αλγόριθμος σε software, χρησιμοποιήσαμε τον βασικά GNU profiler, το gprof.

Τα αποτελέσματα που πήραμε από το gprof είναι τα εξής:

% time	Cumulative seconds	Self seconds	Number of calls	Self ms/call	Total ms/call	Function name
35.86	0.05	0.05	2,400,000	0.02	0.02	FI
35.86	0.10	0.05	800,000	0.06	0.13	FO
14.34	0.12	0.02	100,000	0.20	0.20	KeySchedule
7.17	0.13	0.01	100,000	0.10	1.10	Kasumi
7.17	0.14	0.01	-	-	-	Main
0.00	0.14	0.00	800,000	0.00	0.00	FL

Πίνακας 3.1 Software profiling για τον αλγόριθμο A5/3

Επεξήγηση του παραπάνω πίνακα :

- Η στήλη “% time” αναφέρεται στο ποσοστό, επί του συνολικού χρόνου που κάνει για να εκτελεστεί όλο το πρόγραμμα, που καταναλώνει η συγκεκριμένη συνάρτηση.
- Η στήλη “cumulative seconds” υπολογίζεται από το άθροισμα του χρόνου που καταναλώνει η συγκεκριμένη συνάρτηση μαζί με μας όσες βρίσκονται από πάνω μας στον πίνακα.
- Η στήλη “self seconds” αναφέρεται στον συνολικό χρόνο σε δευτερόλεπτα που καταναλώνει η κάθε συνάρτηση ξεχωριστά.
- Η στήλη “calls” αναφέρεται στον συνολικό αριθμό κλήσεων που δέχεται η κάθε συνάρτηση.
- Η στήλη “self ms/call” αναφέρεται στον μέσο αριθμό από milliseconds που ξοδεύονται στην συγκεκριμένη συνάρτηση για κάθε κλήση μας.
- Η στήλη total ms/call αναφέρεται στον μέσο αριθμό από milliseconds που ξοδεύονται όταν καλείται η συγκεκριμένη συνάρτηση καθώς και όλοι οι απόγονοί μας.

Για να μπορέσουμε να πάρουμε αποτελέσματα από το Gprof, έπρεπε να προσομοιώσουμε τον κώδικά μας ώστε να τρέξει συνολικά 100,000 φορές. Για λιγότερες κλήσεις του προγράμματός μας, το Gprof δεν μπορούσε να μας βγάλει σωστά αποτελέσματα. Ο λόγος είναι ότι ο profiler μας παίρνει ουσιαστικά δείγματα ανά συγκεκριμένες χρονικές στιγμές (sampling) από τον επεξεργαστή για την λειτουργία του. Επειδή όμως το πρόγραμμά μας είναι αρκετά μικρό, τρέχει σε πολύ μικρό χρονικό διάστημα (της τάξης των ns) και κατά συνέπεια ο

profiler δεν μπορεί να κάνει σωστό sampling.

Να αναφέρουμε ότι η συνάρτηση main έχει κενές μας στήλες number of calls, self ms/call, total ms/call καθώς δεν μας ενδιαφέρει να αναλύσουμε την συνάρτηση main. Η συνάρτηση main υλοποιήθηκε μόνο για να μπορεί να τρέξει το πρόγραμμά μας. Μας, αφαιρέσαμε μας μας κλήσεις συστήματος (system calls) που υπήρχαν στο πρόγραμμα (π.χ. printf). Η ορθότητα μας υλοποίησής μας ελέγχθηκε με τα ίδια test data που χρησιμοποιήθηκαν και στην hardware υλοποίηση (αν και ο κώδικας παρέχεται έτσι κι αλλιώς από την 3GPP).

Συνολικά λοιπόν, αναλύοντας τα αποτελέσματα του παραπάνω πίνακα, βλέπουμε ότι χρειάστηκαν 0.14 seconds για να τρέξει ο αλγόριθμος 100,000 φορές.

Συνεπώς, για να τρέξει 1 φορά θα χρειαστεί

$$\frac{0.14}{100{,}000}\text{sec} = 1400\ ns$$

Συγκριτικά με την υλοποίησή μας σε hardware έχουμε :

Χρόνος σε hardware	Χρόνος σε software	Speedup
72.792 ns	1400 ns	19.23

Πίνακας 3.2 Speedup σε hardware υλοποίηση του A5/3

Ο λόγος που υλοποιήσαμε τον αλγόριθμο A5/3 σε software δεν είναι η σύγκριση με την αντίστοιχη υλοποίηση σε hardware (άλλωστε ο αλγόριθμος είναι φτιαγμένος για να δουλεύει σε hardware, τα κινητά δεν χρησιμοποιούν software υλοποίησή του). Ο λόγος είναι ότι προκειμένου να μελετήσουμε τεχνικές επίθεσης στον συγκεκριμένο αλγόριθμο, θα πρέπει να συγκρίνουμε τις τεχνικές αυτές υλοποιημένες σε hardware αλλά και σε software, συνεπώς χρειαζόμαστε τον κρυπταλγόριθμο και στις δύο του μορφές.

3.9 Επαλήθευση των αποτελεσμάτων της υλοποίησης

Προκειμένου να επαληθεύσουμε την ορθότητα της σχεδίασής μας, χρησιμοποιήσαμε τα αποτελέσματα της τεχνικής αναφοράς [4] και συγκεκριμένα του τρίτου κεφαλαίου, “Algorithm A5/3 for GSM”.

Από την αναφορά αυτή παίρνουμε τα εξής αποτελέσματα:

KLEN	K_C	COUNT	BLOCK1	BLOCK2
64	0x2BD6459F82C5BC00	0x24F20F	0x889EEAAF9ED1BA1ABBD8436232E440	0x5CA3406AA244CF69CF047AADA2DF40
64	0x952C49104881FF48	0x061527	0xAB7DB38A573A325DAA76E4CB800A40	0x4C4B594FEA9D00FE8978B7B7BC1080
64	0xEFA8B2229E720C2A	0x33FD3F	0x0E4015755A336469C3DD8680E30340	0x6F10669E2B4E18B042431A28E47F80
64	0x3451F23A43BD2C87	0x0E418C	0x75F7C4C51560905DFBA05E46FB54C0	0x192C95353CDF979E054186DF15BF00
64	0xCAA2639BE82435CF	0x2FF229	0x301437E4D4D6565D4904C631606EC0	0xF0A3B8795E264D3E1A82F684353DC0
64	0x7AE67E87400B9FA6	0x2F24E5	0xF794290FEF643D2EA348A7796A2100	0xCB6FA6C6B8A705AF9FEFE975818500

Πίνακας 3.3 Αποτελέσματα του αλγορίθμου

Στον παραπάνω πίνακα, έχουμε μήκος κλειδιού 64 bits, παρ όλα αυτά, η συγκεκριμένη τιμή διπλασιάζεται ώστε να μας δώσει το τελικό μας κλειδί. Επίσης, η μεταβλητή COUNT που βλέπουμε στον πίνακα είναι ένα μέρος της εισόδου του κρυπταλγόριθμου (μέρος του plaintext δηλαδή) και ο τρόπος που παράγεται το plaintext εξηγείται αναλυτικά στο [4].

Τα αποτελέσματα που παίρνουμε χωρίζονται σε BLOCK1 και BLOCK2 επειδή στην γενικότερη υλοποίηση της συνάρτησης f9, χρησιμοποιούμε πολλά αντίγραφα του KASUMI. Παρ όλα αυτά, ελέγχτηκαν ότι παράγονται σωστά και τα δύο blocks, τόσο με την software υλοποίησή μας όσο και με την hardware.

Έχοντας την υλοποίηση του αλγορίθμου τόσο σε hardware όσο και σε software μπορούμε να προχωρήσουμε στην δημιουργία των πινάκων ουράνιου τόξου όπως θα δούμε στα επόμενα κεφάλαια.

4 Κρυπτανάλυση και πίνακες ουράνιου τόξου

4.1 Γενικά για τους πίνακες ουράνιου τόξου

Ένας πίνακας ουράνιου τόξου είναι ένας προϋπολογισμένος πίνακας που αντιστοιχίζει κωδικούς (passwords) με κατακερματισμένους κωδικούς (hash passwords). Το κύριο χαρακτηριστικό των πινάκων ουράνιου τόξου είναι ότι προσφέρουν μια ανταλλαγή χρόνου-μνήμης (time-memory tradeoff).

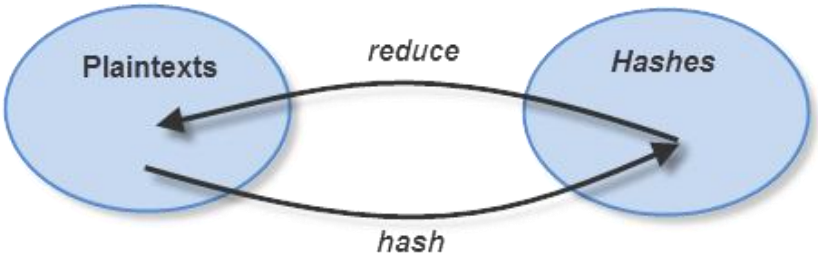
Με τον όρο ανταλλαγή χρόνου-μνήμης εννοούμε ότι, δεδομένου των σημαντικών βελτιώσεων που παρατηρούνται σε υπολογιστική ισχύ αλλά και μνήμη, μπορούμε να επιλέξουμε την κατάλληλη σχέση μεταξύ των δύο αυτών παραμέτρων έτσι ώστε να έχουμε όφελος ως προς την πολυπλοκότητα του προβλήματός μας. Δηλαδή, να έχουμε την δυνατότητα να μειώσουμε το μέγεθος της μνήμης που χρησιμοποιούμε χάνοντας παράλληλα και σε χρόνο εκτέλεσης αλλά και να μπορούμε να μεγαλώσουμε την μνήμη μας, βελτιώνοντας τον χρόνο εκτέλεσης.

Η πρώτη διατύπωση των πινάκων ουράνιου τόξου ανήκει στον Martin Hellman ο οποίος το 1980 παρουσίασε μια κρυπταναλυτική μέθοδο η οποία μπορούσε να εκμεταλλευτεί την ανταλλαγή χρόνου-μνήμης με σκοπό να μειώσει τον συνολικό χρόνο που απαιτούσε η διαδικασία της κρυπτανάλυσης [7]. Ωστόσο, στην τελική τους μορφή με την οποία και θα ασχοληθούμε τους έφερε ο Philippe Oechslin ο οποίος στηριζόμενος στις ήδη σχετικές εργασίες παρουσίασε μια καινούρια δομή για τους πίνακες ουράνιου τόξου με την οποία μπορούσε για παράδειγμα να ανακτήσει ένα κωδικό των MS-Windows μέσα σε 13.6 δευτερόλεπτα με πιθανότητα επιτυχίας 0.99 [1]. Η τεχνική των πινάκων ουράνιου τόξου μπορεί ωστόσο να επεκταθεί και σε άλλους κρυπτογραφικούς αλγόριθμους.

4.2 Αναλυτική περιγραφή παραγωγής πινάκων

Για να παράγουμε έναν τέτοιο πίνακα αρχικά χρειαζόμαστε μια συνάρτηση κατακερματισμού (hash function). Η συνάρτηση αυτή θα δέχεται σαν είσοδο κάποιους τυχαίους κωδικούς που θα παράγουμε εμείς και σαν έξοδο τους κατακερματισμένους κωδικούς. Επίσης, θα δημιουργήσουμε μια συνάρτηση ελάττωσης (reduction function) η οποία θα αντιστοιχίζει κατακερματισμένους κωδικούς σε πραγματικούς κωδικούς.

Σχηματικά δηλαδή θα έχουμε :



Εικόνα *4.2* Διαδικασία hash-reduce

Είναι σημαντικό να τονίσουμε ότι η reduction function κάνει την αντίστροφη διαδικασία από την hash function, δεν είναι όμως και ανάστροφή της. Άλλωστε η ουσία των hash functions είναι ότι σχεδιάζονται για να μην μπορούν να αναστραφούν. Στο παραπάνω σχήμα, το αποτέλεσμα της ελάττωσης του κατακερματισμένου κωδικού σίγουρα δεν είναι το ίδιο με τον αρχικό κωδικό.

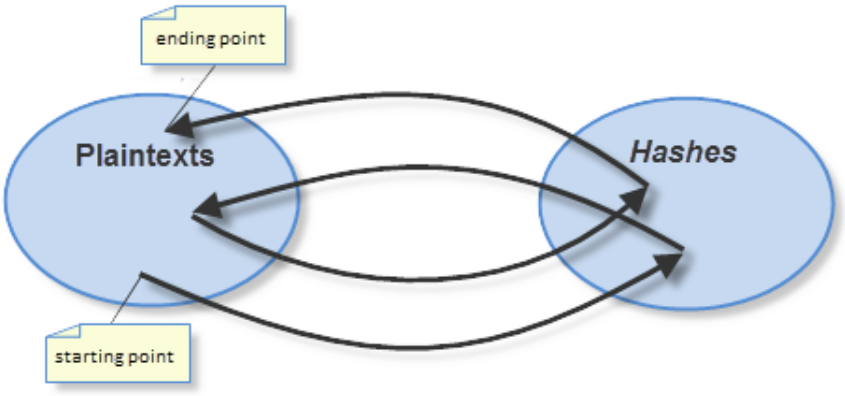
Έστω για παράδειγμα ότι έχουμε το σετ από κωδικούς [0123456789] {6} (δηλαδή έχουμε όλους τους αριθμητικούς συνδυασμούς από κωδικούς μήκους 6). Αν επιλέξουμε σαν hash function τον κρυπταλγόριθμο MD5, ένα παράδειγμα ζεύγους κωδικού-κατακερματισμένου κωδικού θα μπορούσε να είναι το :

MD5(“493823”) → “222f00dc4b7f9131c89cff641d1a8c50”

Η reduction function που θα χρησιμοποιήσουμε δεν χρειάζεται να είναι πολύπλοκη, μπορεί για παράδειγμα απλά να παίρνει τους πρώτους 6 αριθμούς του κατακερματισμένου κωδικού (θυμίζουμε ότι έχουμε δεχτεί σαν περιορισμό ότι το μήκους κλειδιού είναι 6). Κατά συνέπεια, θα έχουμε :

R(“222f00dc4b7f9131c89cff641d1a8c50”) → “222004”

Δημιουργήσαμε λοιπόν έναν καινούριο κωδικό, κατακερματίζοντας τον παλιό κωδικό, αυτός είναι ο σκοπός της reduction function. Επεκτείνοντας την σκέψη αυτή, μπορούμε να εκτελέσουμε πολλές φορές την παραπάνω διαδικασία, δημιουργώντας μια αλυσίδα (rainbow chain) με τον συνεχόμενη εφαρμογή κατακερματισμού και ελάττωσης σε έναν κωδικό.



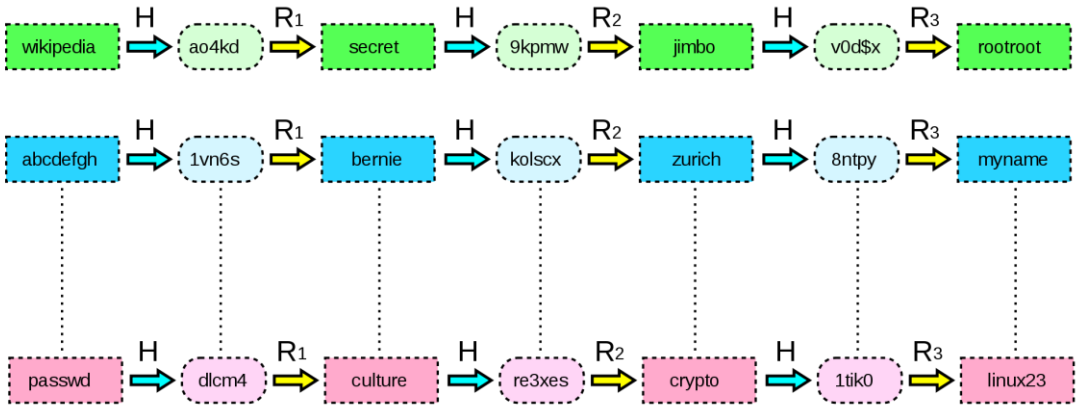
Εικόνα *4.3* Διαδικασία hash-reduce για δημιουργία αλυσίδας

Οι συναρτήσεις κατακερματισμού, όπως και οι συναρτήσεις ελάττωσης είναι μη ανάστροφες. Οι αλυσίδες οι οποίες θα σχηματίσουν τον τελικό μας πίνακα ουράνιου τόξου είναι αλυσίδες από διαδοχικές hash functions και reduction functions στη σειρά. Από το μήκος της αλυσίδας εξαρτάται σε μεγάλο ποσοστό η επιτυχία του αλγορίθμου μας.

Η εκάστοτε αλυσίδα ξεκινάει από έναν τυχαίο κωδικό, τον κατακερματίζει, ελαττώνει τον κατακερματισμένο κωδικό, κατακερματίζει τον καινούριο κωδικό κ.ο.κ. Το μήκος της αλυσίδας είναι σταθερό για όλη την διαδικασία. Το σημαντικό όμως είναι ότι ο τελικός πίνακας ουράνιου τόξου χρειάζεται μόνο τον αρχικό κωδικό και τον τελικό κατακερματισμένο κωδικό, όχι όλους τους ενδιάμεσους που παρήχθησαν (εξηγούμε πιο κάτω τον λόγο). Κατά συνέπεια, μια αλυσίδα από εκατομμύρια ίσως κατακερματισμένους κωδικούς, αναπαριστάται τελικά στον πίνακά μας με ένα μόνο ζεύγος κωδικών.

Έστω ότι η συνάρτηση κατακερματισμού ονομάζεται H και έχουμε ένα συγκεκριμένο αριθμό από κωδικούς, P. Ο στόχος μας είναι να φτιάξουμε έναν πίνακα μέσα στον οποίο θα μπορούμε, δοσμένου ενός κατακερματισμένου κωδικούς h, να βρούμε τον κωδικό p για τον οποίο ισχύει H(p) = h (ή έστω να αποφανθούμε ότι δεν υπάρχει κωδικός για τον οποίο να ισχύει η παραπάνω σχέση). Αν αποθηκεύαμε κάθε ενδιάμεσο κωδικό που παράγεται από την αλυσίδα θα χρειαζόμασταν (Θ(|P|(n)) bits μνήμης όπου n είναι το μέγεθος της εξόδου της συνάρτησης κατακερματισμούς και μπορεί να είναι απαγορευτικό για μεγάλο P.

Θα ονομάσουμε σαν αρχικό σημείο (starting point, SP) τον πρώτο κωδικό και τελικό σημείο (ending point, EP) τον τελευταίο κωδικό.

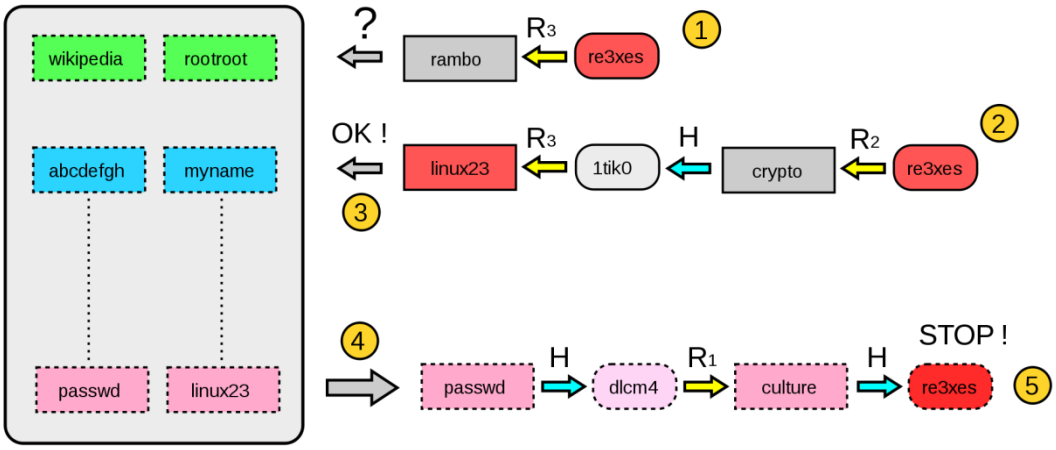


Εικόνα *4.4* Παράδειγμα δημιουργίας αλυσίδων

Στον παραπάνω πίνακα, αν πάρουμε την τελευταία αλυσίδα, το starting point είναι το “passwd” και το ending point το “linux23” και η αλυσίδα έχει μήκος 3.

4.3 Διαδικασία αποκρυπτογράφησης με πίνακες ουράνιου τόξου

Έστω ότι έχουμε δημιουργήσει τον πίνακά μας και θέλουμε, δοσμένου ενός (κατακερματισμένου) κωδικού h, να βρούμε τον κωδικό από τον οποίο προήλθε. Θα περάσουμε αρχικά τον h μέσα από την συνάρτηση R, μετά μέσα από την συνάρτηση H κ.ο.κ. Αν σε αυτή τη διαδικασία δημιουργηθεί ένας κωδικός ο οποίος υπάρχει σαν ending point στον πίνακάς μας, τότε έχουμε βρει τον κωδικό που θέλουμε.



Εικόνα *4.5* Παράδειγμα ανάκτησης κωδικού

Πιο αναλυτικά και με βάση τον παραπάνω πίνακα, ακολουθούμε τα παρακάτω βήματα:

- Έχουμε δοσμένο τον κατακερματισμένο κωδικό h=”re3xes”. Αρχικά, θα εφαρμόσουμε την reduction function η οποία θα μας δώσει το αποτέλεσμα “rambo”.
- Ελέγχουμε αν η λέξη “Rambo” υπάρχει στην στήλη με τα ending points του rainbow table.
- Αν βρούμε την λέξη στον πίνακα, πάμε στο βήμα 5.
- Αν δεν βρούμε την λέξη στον πίνακα, τότε παίρνουμε πάλι τον αρχικό κωδικό (“re3xes”), εφαρμόζουμε την R, έπειτα την H και μετά πάλι την R. Έστω ότι το αποτέλεσμα που πήραμε είναι η λέξη “linux23”. Ψάχνουμε την λέξη αυτή στον πίνακα, στην στήλη με τα ending points. Αν την βρούμε πάμε στο βήμα 5 αλλιώς επαναλαμβάνουμε το βήμα αυτό.
- Πηγαίνουμε στην γραμμή που βρήκαμε τον κωδικό μας και παίρνουμε το starting point. Αν είναι η λέξη “passwd” όπως στην εικόνα, τότε μπορούμε εύκολα να διαπιστώσουμε ότι αν εφαρμόσουμε τις συναρτήσεις H και R όσες φορές τις εφαρμόσαμε στο βήμα 4, θα καταλήξουμε στον κατακερματισμένο κωδικό “re3xes”. Κατά συνέπεια, ο κωδικός ο οποίος προηγείται στην αλυσίδα του “re3xes”, δηλαδή η λέξη “culture” είναι ο κωδικός

τον οποίο ψάχνουμε και η επίθεση ήταν επιτυχής.

Ένα μειονέκτημα των πινάκων αυτών είναι η περίπτωση όπου δύο αλυσίδες, σε κάποιο σημείο του υπολογισμού τους, παράγουν τον ίδιο κατακερματισμένο κωδικό. Σε αυτή την περίπτωση, οι δύο αλυσίδες ενώνονται σε μία ουσιαστικά και στον τελικό μας πίνακα, θα έχουμε δύο ίδια ζεύγη starting-ending point.

Για να αποφευχθεί αυτό, μπορούμε να δημιουργήσουμε πολλές διαφορετικές συναρτήσεις ελάττωσης, οι οποίες θα εφαρμόζονται με συγκεκριμένη σειρά (έτσι ώστε να μπορούμε μετά να ανακτήσουμε τον κωδικό). Για να έχουμε συνένωση δύο αλυσίδων σε αυτή την περίπτωση, θα πρέπει να δημιουργηθούν οι ίδιοι κατακερματισμένοι κωδικοί, στον ίδιο γύρο, πράγμα απίθανο. Ακόμα και στη περίπτωση όπου δημιουργηθούν δύο ίδιες καταχωρήσεις στον τελικό μας πίνακα, τις αφαιρούμε και υπολογίζουμε καινούριες.

Οι πίνακες ουράνιου τόξου είχαν προταθεί από τον Philippe Oechslin και είχαν ως κύριο πλεονέκτημα την ανταλλαγή χρόνου-μνήμης. Η ανταλλαγή αυτή έγκειται στο γεγονός ότι για μεγάλο μήκος αλυσίδας, έχουμε μεγαλύτερο χρόνο υπολογισμού των πινάκων αλλά, χρειαζόμαστε και λιγότερη μνήμη. Αν χρησιμοποιήσουμε μικρότερο μήκος αλυσίδας, θα έχουμε μικρότερο χρόνο υπολογισμό πίνακα αλλά, θα χρειαστούμε μεγαλύτερη μνήμη (για να έχουμε παρόμοια πιθανότητα επιτυχίας εύρεσης κωδικού με βάση τον πίνακα).

4.4 Αντιμετώπιση επιθέσεων με πίνακες ουράνιου τόξου

Μια επίθεση με πίνακες ουράνιου τόξου δεν μπορεί να επιτευχθεί αν ο κρυπτογραφικός αλγόριθμος χρησιμοποιεί την τεχνική του salt. Τα salts είναι κάποιες τυχαίες ακολουθίες από bits που ενώνονται με τα bits εισόδου του κρυπταλγόριθμου έτσι ώστε να καταστήσουν αδύνατη την επίθεση με προϋπολογισμένους πίνακες.

Μια πιθανή έκφραση του salt θα μπορούσε να είναι

Saltedhash (password) = hash (password & salt)

ή

Saltedhash (password) = hash (hash (password) & salt)

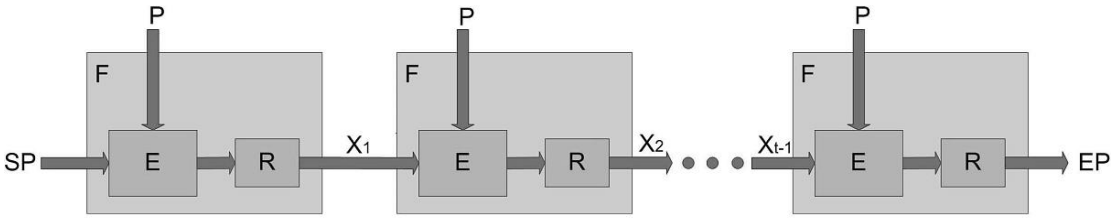
Σε αυτή την περίπτωση λοιπόν, ο επιτιθέμενος θα πρέπει για να δημιουργήσει τους πίνακες ουράνιου τόξου, να τους υπολογίσει για κάθε μία από τις διαφορετικές πιθανές τιμές του salt. Για τον λόγο αυτό, τα salts θα πρέπει να έχουν σχετικά μεγάλο μήκος, σε σύγχρονα λειτουργικά συστήματα Linux χρησιμοποιούνται salts μήκους 48 ή 128 bits.

5 Εφαρμογή των rainbow tables στον αλγόριθμο A5/3

5.1 Γενική μεθοδολογία

Θα προσπαθήσουμε να εφαρμόσουμε τους πίνακες ουράνιου τόξου στον αλγόριθμό μας, για μια συγκεκριμένη είσοδο. Η παράμετρος στην οποία θα επικεντρωθούμε δηλαδή είναι το κλειδί του αλγορίθμου, μήκους 128 bits.

Έστω λοιπόν ότι ο κρυπτογραφικός μας αλγόριθμος είναι η συνάρτηση E του σχήματος, R η reduction function που θα χρησιμοποιήσουμε, και P το σταθερό plaintext που θα έχουμε για όλη την διαδικασία.



Εικόνα *5.1* Διαδικασία δημιουργίας αλυσίδας

Σε πρώτη φάση, επιλέγουμε m διαφορετικά κλειδιά τα οποία θα είναι τα starting points των αλυσίδων. Για να δημιουργήσουμε μια αλυσίδα, πρώτα υπολογίζουμε το $E_{SP}(P)$, δηλαδή το αποτέλεσμα του κρυπταλγόριθμου για είσοδο P και κλειδί κρυπτογράφησης το SP. Το αποτέλεσμα που θα πάρουμε θα είναι ένα κρυπτοκείμενο (cipher text), έστω C. Για να συνεχιστεί η αλυσίδα, το C θα πρέπει να χρησιμοποιηθεί έτσι ώστε να δημιουργήσουμε ένα καινούριο κλειδί. Για τον λόγο αυτό, περνάμε το C μέσα από την reduction function R έτσι ώστε να παράγουμε το καινούριο κλειδί X_1. Κατά σειρά δηλαδή θα έχουμε:

$$R(E_{SP}(P)) = X_1$$

$$R(E_{X1}(P)) = X_2 \quad \text{κ.ο.κ.}$$

Να σημειώσουμε ότι οι reduction functions που χρησιμοποιούνται σε κάθε βήμα είναι διαφορετικές μεταξύ τους έτσι ώστε να μην έχουμε συγχώνευση αλυσίδων σε κανένα σημείο του πίνακά μας.

Για τον συνδυασμό των συναρτήσεων E και R χρησιμοποιείται ο όρος step function. Ουσιαστικά, είναι σαν να ανατροφοδοτούμε την step function τόσες φορές όσο είναι και το μήκος της αλυσίδας μας. Μετά λοιπόν από t εφαρμογές της step function, παίρνουμε το ζεύγος starting-ending point και το αποθηκεύουμε στη μνήμη μας. Ο λόγος των διαφορετικών κλειδιών

που αποθηκεύονται στο τέλος, προς τον αριθμό N (το σύνολο των διαφορετικών κλειδιών) ονομάζεται έκταση (coverage) του πίνακά μας.

Για να διαπιστώσουμε αν ένα κρυπτογραφημένο κείμενο το οποίο δημιουργήθηκε χρησιμοποιώντας ένα κλειδί k, καλύπτεται από έναν πίνακα ουράνιου τόξου, υπολογίζουμε όλες τις αλυσίδες ξεκινώντας από το $R(C)$ και κάνουμε αυτή τη διαδικασία t φορές. Κάθε φορά που δημιουργούμε έναν κωδικό, ελέγχουμε αν αυτός υπάρχει στον πίνακά μας. Αν δεν υπάρχει, τότε υπολογίζουμε το $F(R\ C\ )$, βλέπουμε αν υπάρχει στον πίνακά μας, κ.ο.κ. Γενικά, αν βρούμε σε κάποιον υπολογισμό ένα κλειδί το οποίο να υπάρχει και στον πίνακα ουράνιου τόξου και το κλειδί αυτό έχει προκύψει μετά την ν-οστή φορά εφαρμογής της συνάρτησης F, τότε το

$$F_{t-n-1}\ (\mathrm{SP}) = X_{t-\nu-1}$$

είναι υποψήφιο κλειδί το οποίο όμως θα πρέπει να επαληθευτεί. Αν λοιπόν, για το συγκεκριμένο κλειδί ισχύει επίσης ότι

$$E_{X_{t-\nu-1}}\ (P) = C$$

τότε το ζητούμενο κλειδί έχει βρεθεί και η διαδικασία της αποκρυπτογράφησης ήταν επιτυχής. Αν όχι, τότε θα εφαρμόσουμε την ίδια διαδικασία σε άλλον πίνακα ουράνιου τόξου.

Προκειμένου να πραγματοποιηθεί γρηγορότερα η διαδικασία εύρεσης του κλειδιού μέσα στον πίνακα, φροντίζουμε ώστε να αποθηκεύουμε τα ζεύγη starting-ending points στον πίνακάς μας με βάση το ending point, δημιουργούμε ουσιαστικά ένα hash table. Το μόνο πρόβλημα στην συγκεκριμένη υλοποίηση είναι ότι κάποιες θέσεις στην μνήμη μας μπορεί να μείνουν κενές αλλά αυτό διορθώνεται εύκολα με χρήση πιο αποδοτικών συναρτήσεων κατακερματισμού (hash functions).

5.2 Ανταλλαγή χρόνου-μνήμης (time-memory trade off)

Όπως αναφέρεται και στην εργασία του Oechslin [1], του δημιουργού των πινάκων ουράνιου τόξου, υπάρχουν τρεις βασικές παράμετροι τις οποίες πρέπει να ρυθμίσουμε ώστε να έχουμε σωστό trade-off χρόνου και μνήμης :

- Το μήκος των αλυσίδων, έστω t
- Τον αριθμό των αλυσίδων για κάθε πίνακα, έστω m
- Τον αριθμό των πινάκων που θα παράγουμε, έστω ℓ.

Οι παράμετροι αυτοί, θα πρέπει να ρυθμιστούν και με βάση την συνολική μνήμη που έχουμε διαθέσιμη. Η μνήμη αποτελούσε και το κυριότερο πρόβλημα της εργασίας καθώς το μέγιστο βάθος που μπορούσαμε να έχουμε ήταν 2^{16}. Όλα αυτά επηρεάζουν τον συνολικό ρυθμό επιτυχίας, έστω $P_{success}$.

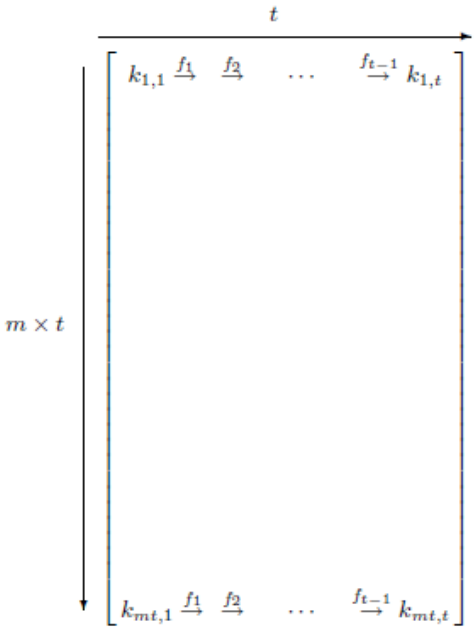
Η συνολική μας μνήμη, έστω M, θα πρέπει να είναι το γινόμενο του συνολικού αριθμού των αλυσίδων που έχουμε επί τον αριθμό των πινάκων που θα δημιουργήσουμε, επί την μνήμη

που καταναλώνουν ένα ζευγάρι από starting και ending points (256 bits). Ο συνολικός χρόνος δίνεται από το γινόμενο του μήκους της αλυσίδας που θα χρησιμοποιήσουμε επί τον αριθμό των πινάκων που θα παράγουμε, επί τον χρόνο που χρειάζεται μία αλυσίδα για να παραχθεί.

$$M = m \times \ell \times m_0$$

$$T = t \times \ell \times t_0$$

Όπως αναφέραμε και πριν, οι συναρτήσεις ελάττωσης (reduction functions) που χρησιμοποιούμε είναι διαφορετικές σε κάθε βήμα υπολογισμού μιας αλυσίδας, έτσι ώστε να αποφύγουμε την πιθανότητα συνένωσης (merge) δύο αλυσίδων. Συνεπώς, η πρώτη συνάρτηση ελάττωσης θα είναι η 1 και η τελευταία η $(t-1)$. Αν για δύο αλυσίδες δεν εμφανιστεί η ίδια τιμή στο ίδιο σημείο υπολογισμού, τότε οι δύο αλυσίδες δεν πρόκειται να συνενωθούν. Σε περίπτωση λοιπόν που παράγουμε για δύο αλυσίδες την ίδια τιμή, η πιθανότητα να συνενωθούν είναι $\frac{1}{t}$. Το t συνήθως είναι της τάξης των εκατοντάδων χιλιάδων οπότε η πιθανότητα είναι σχεδόν μηδενική και άρα αμελητέα.



Εικόνα 5.2 Παράδειγμα πίνακα ουράνιου τόξου

Για να υπολογίσουμε την πιθανότητα επιτυχίας ενός πίνακα έχουμε:

Η πρώτη στήλη του πίνακα θα έχει $m_1 = m$ διαφορετικά κλειδιά. Στην δεύτερη στήλη, τα m_1 διαφορετικά κλειδιά που είχαμε θα παράγουν m_2 διαφορετικά κλειδιά χρησιμοποιώντας keyspace μεγέθους N.

$$m_2 = N\,(1 - \;\; 1 - \frac{1}{N}^{\;\; m1}\;) \;\cong\; N(1 - \; e^{-\,m_1\;N}\,)$$

Εφ όσον κάθε στήλη i έχει m_i διαφορετικά κλειδιά, η πιθανότητα επιτυχίας θα δίνεται από τον τύπο

$$P = 1 - \;\; \prod_{i=1}^{t}(1 - \frac{m_i}{N})$$

Την ίδια λογική μπορούμε να χρησιμοποιήσουμε προκειμένου να υπολογίσουμε τον αριθμό των μη-συγχωνευόμενων αλυσίδων που μπορούν να δημιουργηθούν. Από τη στιγμή που γνωρίζουμε ότι οι συγχωνευόμενες αλυσίδες αναγνωρίζονται λόγω ομοιότητας στο endpoint, είναι προφανές ότι ο αριθμός των διαφορετικών κλειδιών που θα προκύψουν στην τελευταία στήλη m_t θα είναι και ο αριθμός των διαφορετικών αλυσίδων που προέκυψαν. Ο μέγιστος αριθμός διαφορετικών αλυσίδων μπορεί να επιτευχθεί μόνο όταν διαλέγουμε διαφορετικό κλειδί κάθε φορά, μέσα στο key space N, ως starting point. Για τον λόγο αυτό θα έχουμε

$$m_1 = m\,, \quad m_{n+1} = \; N(1 - \; e^{-\,m_n\;N}\,)$$

και κατά συνέπεια η πιθανότητα επιτυχίας θα ισούται με

$$P = 1 - \;\; 1 \; - \frac{m_t}{N}^{\;\; t} \;\; \cong 1 - \; e^{-\,t*m_t\;N}$$

και ο συνολικός χρόνος που χρειάζεται για να δημιουργήσουμε έναν τέτοιο πίνακα ισούται με $N * t$.

Γενικά, όταν εφαρμόζουμε την τεχνική των rainbow tables, συνηθίζουμε να παράγουμε περισσότερους από έναν πίνακες έτσι ώστε να έχουμε καλύτερα αποτελέσματα. Στην εργασία του Oechslin [1] για παράδειγμα τα καλύτερα αποτελέσματα προέκυψαν για 5 πίνακες. Αν έχουμε πολλούς πίνακες, η σχέση της συνολικής πιθανότητας επιτυχίας σε σχέση με την πιθανότητα επιτυχίας ενός πίνακα είναι

$$P \geq 1 - \; (1 - P_{one\;table})^r$$

Ένα συνηθισμένο μέγεθος αλυσίδας για κλειδί μήκους k bits είναι $t = 2^{k\;3}$ το οποίο δίνει πιθανότητα επιτυχίας πάνω από 50%. Η πιθανότητα αυτή εξαρτάται από το πόσες αλυσίδες θα συγχωνευτούν και από το πόσες αλυσίδες θα πέσουν σε κάποιο βρόχο ατέρμονης επανάληψης.

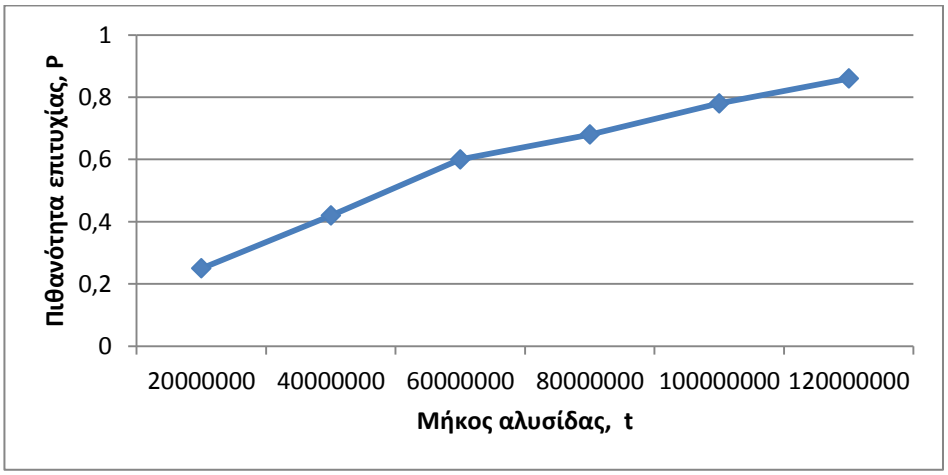
Όπως αναφέραμε, αν και το μήκος του κλειδιού μας είναι 128 bits, προέρχεται από τον διπλασιασμό μιας ποσότητας μεγέθους 64 bits. Συνεπώς το key space που έχουμε, δηλαδή οι δυνατές διαφορετικές τιμές του κλειδιού είναι 2^{64}. Παρόλα αυτά, για δικιάς μας ευκολία,

επιλέγουμε σαν Key space την τιμή 2^{42} καθώς θα ήταν δύσκολο να έχουμε ικανοποιητική πιθανότητα επιτυχίας χρησιμοποιώντας το κανονικό key space.

Ενδεικτικά, παραθέτουμε δύο πίνακες που δείχνουν την πιθανότητα επιτυχίας των πινάκων ουράνιου τόξου καθώς αυξάνεται το μήκος της αλυσίδας.

βάθος μνήμης, m	key space, N	μήκος αλυσίδας, t	Πιθανότητα επιτυχίας, P
65536	2^{42}	5	0
65536	2^{42}	10,000	0.14
65536	2^{42}	51,200,000	0.53
65536	2^{42}	102,400,000	0.78
65536	2^{42}	130,000,000	0.86

Πίνακας 5.1 Υπολογισμός πιθανότητας επιτυχίας για αυξανόμενο μέγεθος αλυσίδας



Πίνακας 5.2 Η πιθανότητα επιτυχίας σε συνάρτηση με το μήκος αλυσίδας

6. Hardware υλοποίηση της δημιουργίας των rainbow tables

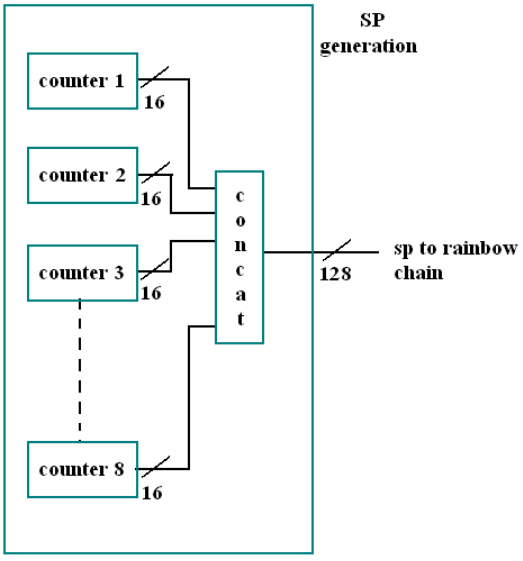
6.1 Γενική περιγραφή της σχεδίασής μας

Προκειμένου να υλοποιήσουμε τους πίνακες ουράνιου τόξου, αρχικά υλοποιήσαμε τον αλγόριθμο κρυπτογράφησης, και έπειτα την λογική όσων χρειάζονται και προαναφέραμε για τους πίνακες δηλαδή, τις hash function, reduction function, έναν κεντρικό controller που θα διαχειρίζεται το όλο σύστημα, έναν SRAM controller για την επικοινωνία με την SRAM και πολλά άλλα. Για την υλοποίηση του αλγορίθμου κρυπτογράφησης αναφερθήκαμε στο δεύτερο κεφάλαιο, οπότε θα παρουσιάσουμε κυρίως την αρχιτεκτονική του συστήματος που δημιουργεί τα rainbow tables.

Χρησιμοποιούμε την αρχιτεκτονική του [5] με τις κατάλληλες αλλαγές έτσι ώστε να μπορεί να εφαρμοστεί στον δικό μας αλγόριθμο κρυπτογράφησης.

6.2 Δημιουργία starting points

Πρώτα απ όλα, έπρεπε να δημιουργήσουμε τυχαία starting points. Τα starting points πρέπει να έχουν μέγεθος 128 bits και να είναι όσο το δυνατόν πιο διαφορετικά μεταξύ τους. Για τον λόγο αυτό χρησιμοποιούμε 8 counters μήκους 16 bits ο καθένας οι οποίοι συνενώνονται για να δημιουργήσουν το εκάστοτε starting point. Να σημειώσουμε ότι ο κάθε μετρητής ξεκινάει από διαφορετική, τυχαία ρίζα (seed) έτσι ώστε να δημιουργηθούν όσο το δυνατόν πιο τυχαία κλειδιά.



Εικόνα *6.1* Starting points module

Επειδή μας ενδιαφέρει η ταχύτητα παραγωγής των πινάκων, είναι σημαντικό τα starting points να παράγονται μέσα στην συνολική αρχιτεκτονική μας και όχι να φορτώνονται από κάποια εξωτερική μνήμη.

6.3 Δημιουργία αλυσίδας

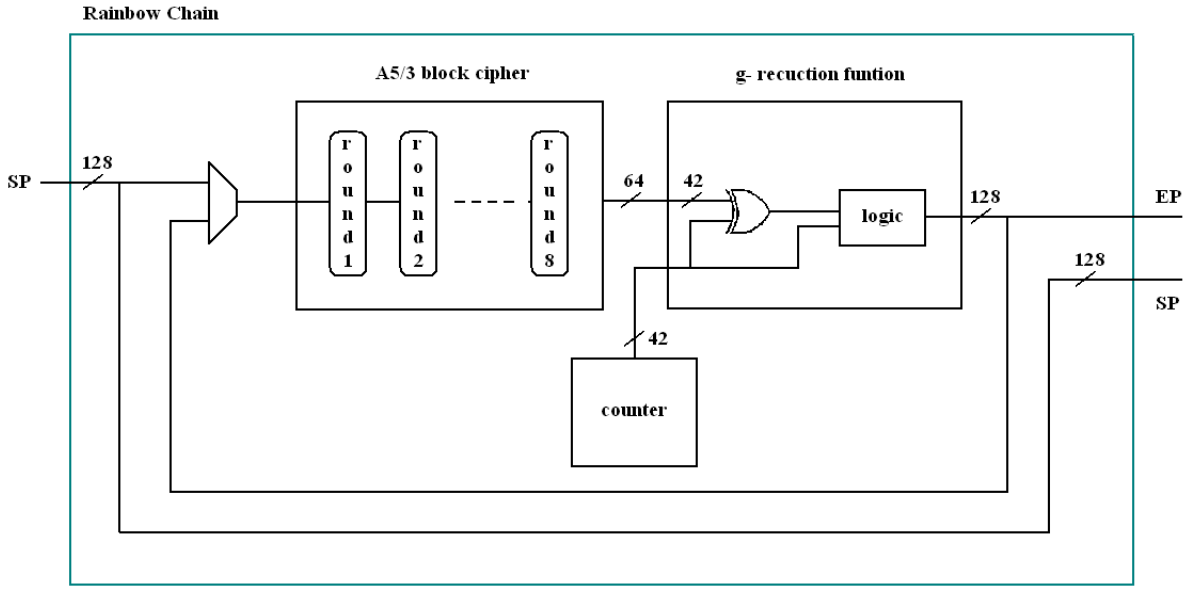
Το κλειδί λοιπόν που παράγεται, θα πρέπει να εισαχθεί στον αλγόριθμο κρυπτογράφησης και, με βάση ένα σταθερό plaintext που χρησιμοποιούμε, θα παράγουμε ένα cipher text. Επειδή το μήκος της αλυσίδας που θα δημιουργήσουμε τελικά είναι σίγουρα μεγαλύτερο από 1, στην είσοδο του κλειδιού του κρυπταλγόριθμου θα έχουμε έναν πολυπλέκτη ο οποίος θα επιλέγει μεταξύ του starting point και του αποτελέσματος της reduction function. Αν θέλουμε να το δούμε σε κύκλους ρολογιού και έχοντας να υπολογίσουμε μια αλυσίδα μήκους 100, από τη στιγμή που παράγεται ένα starting point, θα περάσουν 100*8 = 800 κύκλοι ρολογιού μέχρι να επιλέξουμε το επόμενο starting point.

Η βασική συνάρτηση που πρέπει να δημιουργήσουμε είναι η reduction function. Υπάρχουν πολλοί τρόποι υλοποίησής της, θα πρέπει να λάβουμε όμως υπ όψιν μας ότι η reduction function πρέπει να έχει μικρή πολυπλοκότητα. Για τον λόγο αυτό επιλέγουμε να χρησιμοποιήσουμε την πύλη XOR.

Τα δεδομένα που παίρνουμε από τον κρυπτογραφικό αλγόριθμο (cipher text, μήκους 64 bits) πρέπει να μετατραπούν σε 128-bits ποσότητα έτσι ώστε να διοχετευτούν πάλι στην είσοδο του αλγορίθμου. Για τον λόγο αυτό, γίνονται XOR με την τιμή ενός counter και έπειτα συνενώνονται με μερικά από τα bits αυτού του counter έτσι ώστε να μας δώσουν την ποσότητα που θέλουμε.

Όπως είχαμε αναφέρει και στο κεφάλαιο 4, η reduction function θα πρέπει να είναι διαφορετική κάθε φορά που θέλουμε να κατακερματίσουμε ένα cipher text έτσι ώστε να ελαχιστοποιήσουμε την πιθανότητα να υπάρξουν ίδιες αλυσίδες ουράνιου τόξου. Αυτό το πετυχαίνουμε με τον μετρητή μήκους 42 bits, ο οποίος, κάθε φορά που δημιουργείται ένα καινούριο cipher text, αλλάζει τιμή. Ουσιαστικά, για κάθε διαφορετική τιμή του συγκεκριμένου μετρητή πετυχαίνουμε να έχουμε διαφορετική reduction function.Τα SP και EP, θα μεταφερθούν για εγγραφή στο module EP_mem.

Στην συγκεκριμένη σχεδίαση υπάρχουν και πολλά σήματα ελέγχου τα οποία δεν σημειώνουμε στο σχήμα έτσι ώστε να μην γίνει πολύπλοκο. Δηλαδή, υπάρχει σήμα ενεργοποίησης του αλγορίθμου κρυπτογράφησης, σήμα ενεργοποίησης του μετρητή μας και σαφώς και ένα σήμα select στον πολυπλέκτη έτσι ώστε να επιλέγει τα δεδομένα που θέλει κάθε φορά.



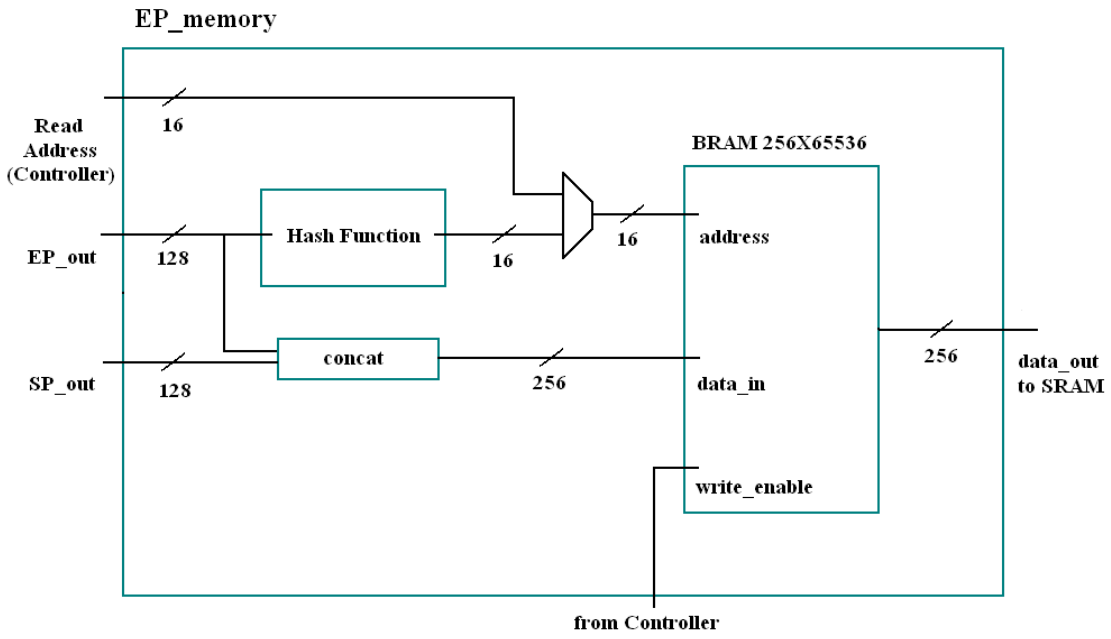
Εικόνα *6.2* Rainbow Chain module

6.4 Αποθήκευση αλυσίδων στη μνήμη

Στο module EP_memory παίρνουμε τα SP και EP που παρήχθησαν από το rainbow chain, και τα αποθηκεύουμε σε μια block ram. Η bram που χρησιμοποιήσαμε ήταν ένα από τα προβλήματά μας καθώς δεν μπορούσε να είναι μεγάλη σε χωρητικότητα. Δεδομένου του μήκους κλειδιού, το μήκος της κάθε γραμμής της μνήμης μας θα πρέπει να είναι 256 bits. Το μέγιστο βάθος μνήμης που μπορούσαμε να παράγουμε (για μια Virtex5 XC5VLX330T) ήταν 65536. Η μνήμη αυτή είναι και ο τελικός μας πίνακας.

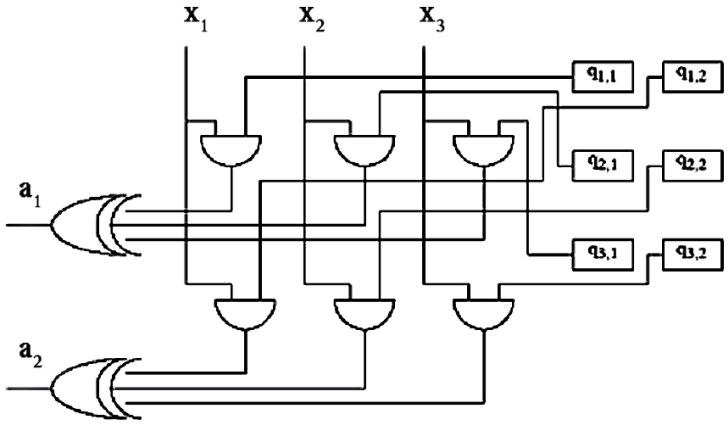
Μία ακόμη παράμετρος που έπρεπε να λάβουμε υπ όψιν είναι ότι οι αλυσίδες που δημιουργούμε δεν πρέπει να γράφονται σειριακά αλλά με βάση το EP (έτσι ώστε κατά την διαδικασία της αποκρυπτογράφησης να μην χρειάζεται να ανατρέχουμε σε όλο τον πίνακα). Χρειαζόμαστε λοιπόν μια hash function η οποία θα δημιουργεί διευθύνσεις εγγραφής για τα δεδομένα στην block ram. Τα δεδομένα που γράφονται στην block ram στέλνονται προς εγγραφή στην SRAM.

Όπως αναφέραμε και προηγουμένως η hash function έχει το μειονέκτημα της μη χρησιμοποίησης κάποιων θέσεων στη μνήμη μας, παρ όλα αυτά θεωρείται βασική στο σύστημά μας καθώς μια εξονυχιστική αναζήτηση στον πίνακα θα ήταν χρονοβόρα. Γενικά διαπιστώσαμε αρκετά καλή διασπορά των αποτελεσμάτων στην μνήμη μας, το ποσοστό των κενών θέσεων που έμεναν ήταν συνήθως λιγότερο από το 1% της μνήμης. Θεωρείται κρίσιμη η επιλογή της hash function για υλοποίηση σε hardware καθώς οι πίνακες πρέπει να είναι ταξινομημένοι και είναι προτιμότερο να διαλέξουμε την καλή λύση από άποψη ταχύτητας.



Εικόνα *6.3* EP_memory module

Το σήμα read_address που φαίνεται στην εικόνα 5.3 ελέγχεται από τον Controller. Επιλέγεται να περάσει από τον πολυπλέκτη μόνο όταν έχει γραφτεί μια αλυσίδα στον πίνακά μας και θέλουμε να την διαβάσουμε για να την γράψουμε στην SRAM. Επίσης, το σήμα που ενεργοποιεί την εγγραφή στην Bram έρχεται και αυτό από τον controller και ενεργοποιείται με μία συγκεκριμένη περίοδο η οποία ισούται με 8 X (μήκος αλυσίδας). Το 8 προκύπτει από τους γύρους του αλγορίθμου A5/3.



Εικόνα 6.*4* Hash function του module EP_memory

Η hash function που φαίνεται στην εικόνα 5.4 μας βοηθάει ώστε να αντιστοιχήσουμε 128 bits του ending point σε μια διεύθυνση μήκους 16 bits. Τα x_1, x_2, x_3 είναι οι είσοδοι της hash function, τα q είναι αρχικοποιημένοι πίνακες βάση των οποίων γίνεται η αντιστοίχηση και τα a_1, a_2 οι έξοδοι της hash function.

Όλη η διαδικασία της παραγωγής του πίνακα ελέγχεται από έναν κεντρικό controller. Ο controller αυτός αναλαμβάνει να ενεργοποιήσει τις κατάλληλες χρονικές στιγμές, τα κατάλληλα σήματα ελέγχου, π.χ. τα select των πολυπλεκτών, το σήμα ενεργοποίησης εγγραφής στην μνήμη και άλλα. Ο κεντρικός controller αναλαμβάνει επίσης να ενημερώσει τον SRAM controller όταν θα ξεκινήσει η διαδικασία εγγραφής στην SRAM.

6.5 Μεταφορά των αλυσίδων στην SRAM

Προκειμένου να μπορούν τα rainbow tables μας να αποθηκευτούν στην SRAM, χρειάστηκε να υλοποιήσουμε την διεπαφή με την οποία θα επικοινωνήσει το module που παράγει τα rainbow tables με την SRAM της CYPRESS [19].

Κάθε φορά που δημιουργείται ένας συνδυασμός από starting και ending point, φροντίζουμε να το διαβάζουμε από την BRAM στην οποία αποθηκεύεται αρχικά, και μέσω του SRAM controller, διοχετεύεται προς την SRAM. Ουσιαστικά, από την στιγμή που δημιουργείται το πρώτο ζεύγος starting-ending point, ενεργοποιείται ο Sram controller ο οποίος παίρνει δεδομένα από την BRAM και αναλαμβάνει να γραφτούν στην Sram.

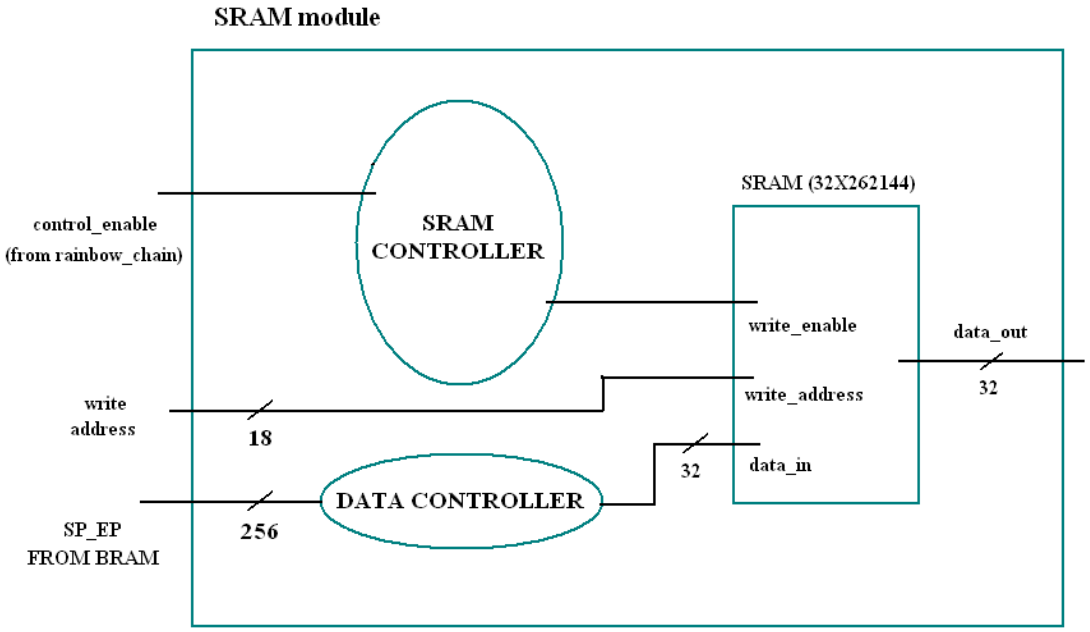
Το σχήμα επικοινωνίας φαίνεται παρακάτω. Λόγω των περιορισμών στο μέγεθος της SRAM (16-bit μέγεθος γραμμής), χωρίζουμε την ποσότητα των 256-bits που πρέπει να γράψουμε σε 4 λέξεις μεγέθους 64 bits, δηλαδή χρησιμοποιούμε τέσσερα αντίγραφα της SRAM που έχουμε ώστε να προκύψει μια καινούρια τετραπλάσιου μεγέθους.

Κατά συνέπεια η μνήμη που δημιουργούμε έχει μήκος γραμμής 64 bits και βάθος $2^{18} = 262144$. Αυτό σημαίνει ότι το σήμα με το οποίο θα διευθυνσιοδοτήσουμε την μνήμη μας θα είναι μήκους 18 bits. Όμως, αυτό το μήκος διεύθυνσης είναι ασυνεπές σε σχέση με το μήκος διεύθυνσης που έχουμε στην BRAM (16 bits), η ασυνέπεια αυτή προκύπτει από το μικρότερο μήκος λέξης. Λόγω μικρότερου μήκους λέξης λοιπόν, προκειμένου να γραφτεί ένα ζευγάρι starting-ending point, θα χρειαστεί να κάνουμε 4 συνεχόμενες εγγραφές στην SRAM.

Για να έχουμε όμως σωστή διεύθυνση εγγραφής, καθώς θέλουμε το rainbow table που παράγουμε να είναι ταξινομημένο με βάση το ending point, παίρνουμε την διεύθυνση που παράγεται από την hash function και την συμπληρώνουμε με δύο bits σαν LSB. Ουσιαστικά αυτά τα δύο bits δεν επηρεάζουν την διεύθυνσή μας, καθώς οι 4 συνεχόμενες εγγραφές που θα πραγματοποιήσουμε (για ένα ζευγάρι starting-ending point) θα έχουν διαφορετικά μόνο τα δύο LSB. Για παράδειγμα, αν η διεύθυνση που παράγεται από την hash function είναι η ”0010010010010010” τότε η διευθύνσεις που θα κάνουμε τις εγγραφές στην SRAM θα είναι οι “0010010010010010 00”, “0010010010010010 01”, “0010010010010010 10”, “0010010010010010 11”. Με αυτόν τον τρόπο επιτυγχάνουμε την σωστή ταξινόμηση των πινάκων μας στην SRAM.

Επίσης, τα σήματα ελέγχου της SRAM φαίνονται να είναι συνολικά 2 στην εικόνα 5.5 (write_enable, read_enable) αλλά στην πραγματικότητα είναι πέντε. Τα σήματα αυτά τα αναλύουμε στην επόμενη ενότητα και ουσιαστικά οι τιμές τους εξαρτώνται από τα σήματα

write_enable και read_enable.



Εικόνα 6.5 SRAM module

6.6 Static RAM module

Το μοντέλο της στατικής μνήμης RAM που χρησιμοποιούμε είναι το CY7C1041BN της Cypress. Η μνήμη είναι οργανωμένη σε 262144 λέξεις των 16 bits η καθεμία. Τα σήματα που καθορίζουν την λειτουργία της μνήμης (ανάγνωση, εγγραφή, αδρανής) είναι τα παρακάτω (όλα είναι μεγέθους 1 bit) :

Chip Enable (CE)

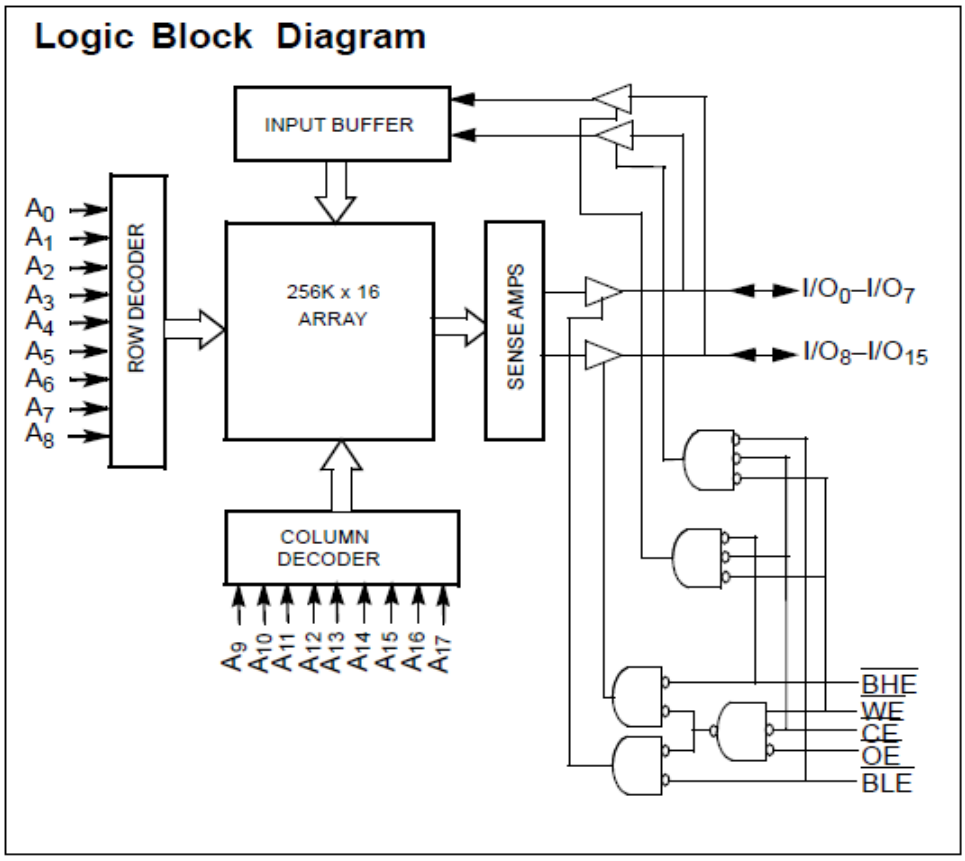
Write Enable (WE)

Byte High Enable (BHE)

Byte Low Enable (BLE)

Output Enable (OE)

Το λογικό διάγραμμα της μνήμης μας είναι το εξής:



Εικόνα 6.6 **SRAM Block Diagram**

Η εγγραφή στην μνήμη επιτυγχάνεται θέτοντας τα σήματα chip enable CE και write enable WE στην τιμή '0'. Αν και το byte low enable, BLE ή το byte high enable, BHE) έχει την τιμή '0' τότε, τα δεδομένα των pins εισόδου-εξόδου IO_0 *έως* IO_7 γράφονται στην διεύθυνση που ορίζουν τα pins διεύθυνσης A_0 *έως* A_{17} .

Η ανάγνωση από την μνήμη επιτυγχάνεται θέτοντας στα σήματα chip enable CE και output enable OE την τιμή '0' και στο write enable WE την τιμή '1'. Αν και το byte low enable, BLE (ή το byte high enable, BHE) έχει την τιμή '0' τότε τα δεδομένα στων οποίων την διεύθυνση δείχνουν τα pins διεύθυνσης, εμφανίζονται στα pins εισόδου-εξόδου. Είναι προφανές ότι δεν μπορούμε να θέσουμε ταυτόχρονα την μνήμη μας σε διαδικασία ανάγνωσης και εγγραφής.

Τα pins εισόδου/εξόδου τίθενται σε κατάσταση υψηλής σύνθετης αντίστασης όταν η μνήμη δεν είναι επιλεγμένη (δηλαδή το CE έχει την τιμή '1') ή όταν οι έξοδοι δεν είναι επιλεγμένοι (δηλαδή το OE έχει την τιμή '1') ή όταν τα BHE και BLE είναι απενεργοποιημένα αλλά και κατά τη διάρκεια μιας εγγραφής ($CE =' 0'$, $WE = '0'$.

Όλες οι λειτουργίες της μνήμης που προαναφέραμε φαίνονται και από τον παρακάτω πίνακα αληθείας

Truth Table

$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	$\overline{BLE}$	$\overline{BHE}$	I/O_0–I/O_7	I/O_8–I/O_{15}	Mode	Power
H	X	X	X	X	High Z	High Z	Power Down	Standby (I_{SB})
L	L	H	L	L	Data Out	Data Out	Read All bits	Active (I_{CC})
L	L	H	L	H	Data Out	High Z	Read Lower bits only	Active (I_{CC})
L	L	H	H	L	High Z	Data Out	Read Upper bits only	Active (I_{CC})
L	X	L	L	L	Data In	Data In	Write All bits	Active (I_{CC})
L	X	L	L	H	Data In	High Z	Write Lower bits only	Active (I_{CC})
L	X	L	H	L	High Z	Data In	Write Upper bits only	Active (I_{CC})
L	H	H	X	X	High Z	High Z	Selected, Outputs Disabled	Active (I_{CC})

Πίνακας 6.1 SRAM truth table

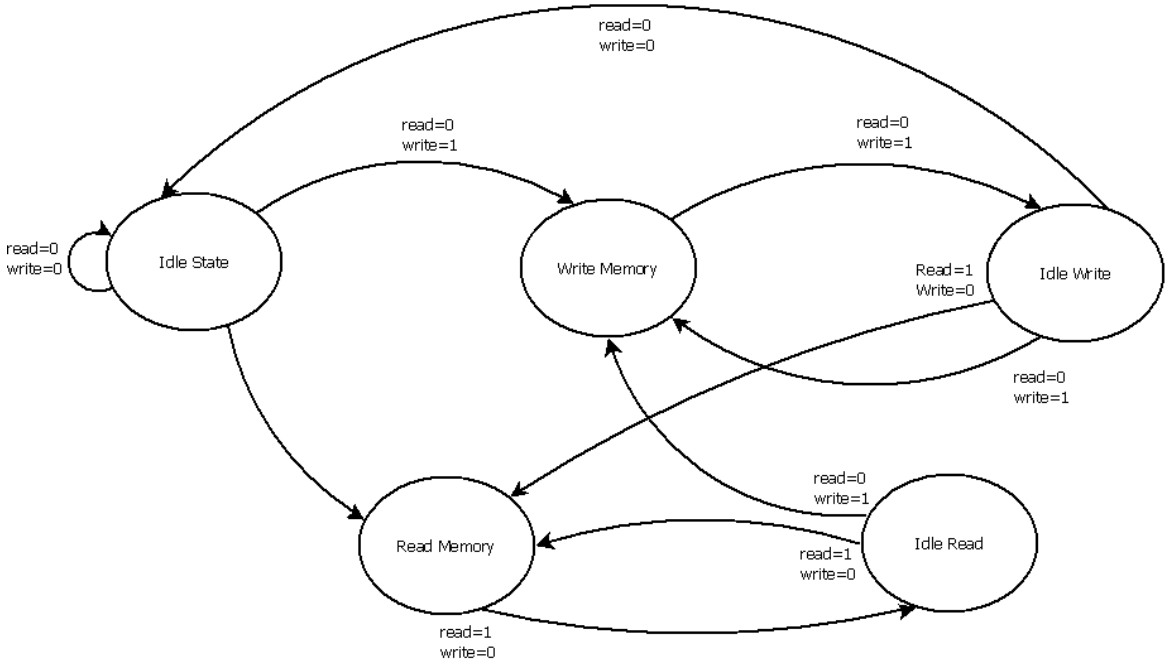
Να σημειώσουμε ότι λόγω του μικρού μεγέθους γραμμής που έχει η μνήμη, χρησιμοποιήσαμε στην σχεδίασή μας τέσσερα αντίγραφα της μνήμης σε σύνδεση έτσι ώστε να έχουμε μία μνήμη με το ίδιο βάθος (262,144) αλλά τετραπλάσιο μήκος λέξης (64 bits). Συγκεκριμένα, ενώσαμε τα σήματα διεύθυνσης των μνημών (για να έχουμε το ίδιο βάθος μνήμης) αλλά και τα σήματα ελέγχου τους.

Επίσης αν και τα σήματα που μας έρχονται από τον SRAM controller είναι τα write_enable και read_enable, φροντίζουμε να τα μεταφράζουμε στα σήματα που χρησιμοποιεί η SRAM (CE, OE, WE, BLE, BHE). Για λόγους απλοποίησης αυτό δεν φαίνεται στο σχήμα.

6.7 SRAM Controller architecture

Προκειμένου να μπορούμε να γράφουμε και να διαβάζουμε από την SRAM χρησιμοποιούμε δύο σήματα, read και write. Οι καταστάσεις της μηχανής πεπερασμένων καταστάσεων που ελέγχει την SRAM φαίνονται παρακάτω.

Ο συγκεκριμένος controller περιμένει να δει κάποια αλλαγή στα σήματα Read ή Write. Στην κατάσταση όπου γράφουμε στην μνήμη ενεργοποιούμε τα κατάλληλα σήματα της SRAM όπως αναφέρθηκαν στο προηγούμενο κεφάλαιο προκειμένου να πραγματοποιηθεί η εγγραφή. Να σημειώσουμε ότι στην συγκεκριμένη εργασία, αν και υλοποιήθηκαν οι καταστάσεις ανάγνωσης από την SRAM, δεν τις χρησιμοποιήσαμε καθώς οι καταστάσεις αυτές μπορούν να χρησιμοποιηθούν για την διαδικασία της εύρεσης ενός κλειδιού μέσα στον πίνακα. Στην συγκεκριμένη εργασία πραγματοποιούμε την διαδικασία δημιουργίας των πινάκων ουράνιου τόξου.



Εικόνα 6.7 SRAM Controller Architecture

Όταν ο SRAM controller δει ότι έχει μια καινούρια λέξη για να γράψει, τότε πραγματοποιεί 4 φορές την εναλλαγή μεταξύ των καταστάσεων write memory – idle write. Σε κάθε εναλλαγή γράφει και από μια 64-bits ποσότητα, στην κατάλληλη διεύθυνση που έχει προκύψει από την διεύθυνση που μας έδωσε η hash function με zero filling κατά δύο bits. Μετά την διαδικασία της εγγραφής οδηγούμε τον controller μας στις καταστάσεις ανάγνωσης της SRAM, δηλαδή στις read memory και idle read. Τις τιμές που διαβάζουμε από την SRAM δεν τις χρησιμοποιούμε, αφήνεται για μελλοντική δουλειά η μεταφορά τους σε υπολογιστή, σε σκληρό δίσκο. Μετά από την ανάγνωση από την SRAM, ο controller μας επιστρέφει στην αρχική του κατάσταση (idle state) και περιμένει το επόμενο ζεύγος starting-ending point από την διαδικασία παραγωγής πινάκων.

Επίσης, λόγω του γεγονότος ότι η συνολική αρχιτεκτονική είναι μικρή σε μέγεθος για την συνολική χωρητικότητα της FPGA, θα δούμε στο επόμενο κεφάλαιο ότι προχωρήσαμε σε παραλληλισμό της σχεδίασής μας. Κατά συνέπεια, και ο SRAM Controller έπρεπε να τροποποιηθεί κατάλληλα ώστε να παίρνει δεδομένα από 2,16,32 και 64 διαφορετικά μηχανήματα παραγωγής πινάκων ουράνιου τόξου.

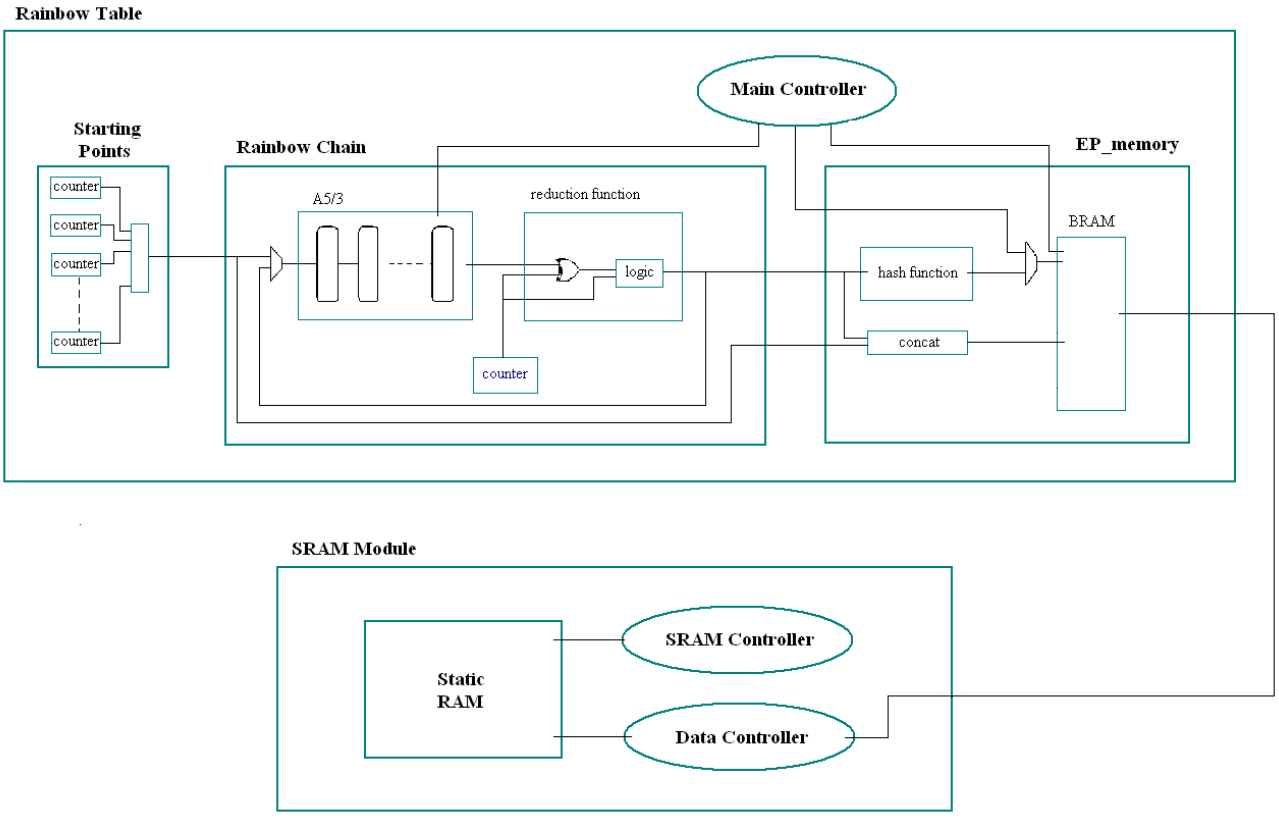
Έστω για παράδειγμα ότι έχουμε δύο παράλληλα μηχανήματα παραγωγής πινάκων ουράνιου τόξου, το αποτέλεσμα είναι ότι ανά συγκεκριμένη χρονική περίοδο θα παράγονται και καινούρια ζεύγη starting-ending points. Επειδή θα έχουμε δύο τέτοια ζευγάρια, έχουμε υλοποιήσει έναν καταχωρητή ο οποίος κρατάει την τιμή του δεύτερου ζευγαριού και το γράφει στην SRAM μόλις τελειώσει η εγγραφή του πρώτου. Προβλήματα συγχρονισμού δεν υπάρχουν καθώς τα επόμενα ζεύγη που θα προκύψουν θα καθυστερήσουν πολύ περισσότερο από όσο διαρκεί μια εγγραφή στην SRAM. Ο λόγος είναι το μεγάλο μήκος αλυσίδας που θα πρέπει να χρησιμοποιηθεί αν θέλουμε να έχουμε σχετικά καλή πιθανότητα επιτυχίας στους πίνακές μας.

Λόγω του μεγάλου μήκους αλυσίδας μπορεί να καθυστερήσει και κατά εκατομμύρια κύκλους να μας έρθουν τα επόμενα ζευγάρια κλειδιών.

Έχουμε υλοποιήσει σε κώδικα VHDL την περίπτωση του SRAM Controller για δύο μηχανήματα, είναι προφανής ο τρόπος λειτουργίας για περισσότερα. Συνολικά, η επιπλέον λογική που προσθέσαμε στον sram controller ώστε να είναι λειτουργικός για παραλληλοποιημένη αρχιτεκτονική είχε μικρό κόστος στο συνολικό μέγεθος της αρχιτεκτονικής μας.

6.8 Συνολική αρχιτεκτονική του συστήματός μας

Σύμφωνα με τα όσα αναφέραμε στις προηγούμενες ενότητες, η συνολική αρχιτεκτονική του συστήματός μας είναι η εξής:



Εικόνα 6.8 Συνολική αρχιτεκτονική του συστήματος

Χρησιμοποιήσαμε την αρχιτεκτονική αυτή προκειμένου να παράγουμε τους πίνακές μας και να συγκρίνουμε τα αποτελέσματα σε χρόνο σε σχέση με την software υλοποίηση των πινάκων ουράνιου τόξου.

Η γενική δομή ακολουθίας που χρησιμοποιούμε είναι:

- Παράγουμε ένα starting point, το αρχικό μας κλειδί

- Από το κλειδί αυτό δημιουργούμε έναν κατακερματισμένο κωδικό
- Δημιουργούμε ένα καινούριο κλειδί μέσω της συνάρτησης ελάττωσης
- Επιστρέφουμε στο δεύτερο βήμα και επαναλαμβάνουμε τόσες φορές όσο είναι και το μήκος της αλυσίδας που χρησιμοποιούμε
- Γράφουμε το starting και το ending point που παρήχθη στην Bram.
- Ταυτόχρονα, διαβάζουμε από την Bram τα δεδομένα που μόλις γράφτηκαν, τα μεταφέρουμε προς εγγραφή στην Sram και συνεχίζουμε ξανά από το $1^{ο}$ βήμα.

Πραγματοποιήσαμε πολλά test benches τα οποία έδειχναν την ορθή λειτουργία του όλου συστήματος. Πέρα από την ορθή λειτουργία του κρυπτογραφικού αλγόριθμου ελέγχτηκαν οι σωστές εγγραφές στις μνήμες καθώς και η διασπορά των εγγραφών. Σε γενικές γραμμές υπήρχαν πολύ λίγες θέσεις που έμεναν κενές κατά τη διάρκεια της παραγωγής των πινάκων.

7 Αποτελέσματα, συγκρίσεις και μελλοντική δουλειά

7.1 Αποτελέσματα και συγκρίσεις με την υλοποίηση σε software

Τα αποτελέσματα που πήραμε για το synthesize της αρχιτεκτονικής μας ήταν τα εξής

Slice utilization	Minimum period	Maximum frequency
2,195/207,360 1%	10.882 ns	91.896 MHz

Πίνακας 7.1 Synthesize results ενός μηχανήματος

Η αρχιτεκτονική αυτή περιλαμβάνει την διαδικασία παραγωγής των rainbow tables καθώς και τον sram controller που θα αναλάβει να μεταφέρει τις αλυσίδες που δημιουργούνται στην sram. Να σημειώσουμε ότι ο sram controller λειτουργεί με διαφορετικό σήμα ρολογιού από την υπόλοιπη σχεδίασή μας καθώς παίρνει το σήμα ρολογιού που έχει η sram (περίπου 250 MHz). Ωστόσο, εμάς θα μας απασχολήσει περισσότερο το ρολόι της υπόλοιπης σχεδίασης καθώς αυτό καθορίζει την ταχύτητα παραγωγής των πινάκων. Ακόμα και σε παραλληλοποιημένες αρχιτεκτονικές, η διαδικασία εγγραφής στην sram είναι πολύ γρηγορότερη από την διαδικασία παραγωγής των πινάκων

Χρησιμοποιήσαμε κάποιες ενδεικτικές τιμές για τις βασικές παραμέτρους που μας χρειάζονται για τους πίνακες ουράνιου τόξου, δηλαδή το μήκος της αλυσίδας και τον αριθμό αλυσίδων για κάθε πίνακα που θα έχουμε. Οι τιμές αυτές επιλέχτηκαν κατάλληλα έτσι ώστε να έχουμε μία καλή πιθανότητα επιτυχίας εύρεσης ενός τυχαίου κλειδιού μέσα στον πίνακα. Θυμίζουμε ότι η πιθανότητα επιτυχίας ενός πίνακα δίνεται από τον τύπο

$$P = 1 - \left(1 - \frac{m_t}{N}\right)^t \cong 1 - e^{-\frac{t * m_t}{N}}$$

Είναι εμφανές από τον παραπάνω τύπο ότι για να έχουμε μεγάλη πιθανότητα επιτυχίας θα πρέπει να έχουμε πολύ μεγάλη τιμή για το μήκος αλυσίδας. Στην υλοποίησή μας, έχουμε κάνει την παραδοχή ότι το key space είναι 2^{42}, ενώ στην πραγματικότητα είναι 2^{64}. Το τελευταίο προκύπτει από το μήκος κλειδιού που είναι 128 bits όμως, προέρχεται από διπλασιασμό μιας 64-bits ποσότητας. Ο λόγος που το κάναμε αυτό είναι για δικιά μας ευκολία, καθώς για μεγάλα key space, η πιθανότητα επιτυχίας μειώνεται κατά πολύ. Ενδεικτικά να αναφέρουμε ότι για key space π.χ. 2^{50} και για το μέγιστο βάθος μνήμης που μπορούμε να έχουμε σε hardware, η πιθανότητα επιτυχίας μειώνεται στο 0,01. Το πρόβλημα αυτό προκύπτει από την μικρή μνήμη που διαθέτουμε η οποία είναι και το bottleneck της υλοποίησής μας. Όσο μεγαλύτερη είναι η μνήμη, τόσο μεγαλύτερους πίνακες μπορούμε να αποθηκεύσουμε και, σε συνδυασμό με αύξηση στο μήκος αλυσίδας μπορούμε να πετυχαίνουμε καλύτερη πιθανότητα επιτυχίας των πινάκων. Βέβαια, η αύξηση του μήκους αλυσίδας θα αύξανε κατά πολύ και τον χρόνο υπολογισμού για έναν πίνακα, οπότε είναι κρίσιμες οι επιλογές που θα κάνουμε για τις παραπάνω παραμέτρους.

Στον πίνακα 6.2 παραθέτουμε ενδεικτικές τιμές για διάφορες τιμές των παραμέτρων

(μήκος αλυσίδας, βάθος μνήμης) και τα αποτελέσματα που παίρνουμε για τον χρόνο υπολογισμού των πινάκων καθώς και την πιθανότητα επιτυχίας εύρεσης ενός κλειδιού μέσα στον πίνακα.

Κύκλοι μίας επανάληψης	Κύκλοι μίας επανάληψης (ns)	Αριθμός αλυσίδων για κάθε πίνακα, m	Μήκος αλυσίδας, t	Clock cycle (ns)
8	87.056	131,072	66,000,000	10.882
8	87.056	16,384	520,000,000	10.882
8	87.056	32,768	256,000,000	10.882
8	87.056	65,536	100,000,000	10.882
8	87.056	65,536	30,000,000	10.882

Πίνακας 7.2 Χρόνοι δημιουργίας rainbow tables για ένα μηχάνημα

Key space, N	Πιθανότητα επιτυχίας, P	Χρόνος υπολογισμού αλυσίδας (ns)	Χρόνος υπολογισμού αλυσίδας (min)	Χρόνος υπολογισμού πίνακα (μέρες)
2^{42}	0.86	5,745,696,000	0.09	8.71
2^{42}	0.86	45,269,120,000	0.75	8.58
2^{42}	0.86	22,286,336,000	0.37	8.45
2^{42}	0.77	8,705,600,000	0.14	6.6
2^{42}	0.36	2,611,680,000	0.04	1.98

Πίνακας 7.3 Χρόνοι δημιουργίας rainbow tables για ένα μηχάνημα

Προκειμένου να υπολογίσουμε τους συνολικούς χρόνους παραγωγής των πινάκων χρησιμοποιήσαμε τους εξής τύπους :

χρόνος υπολογισμού πίνακα = (χρόνος υπολογισμού αλυσίδας) X (αριθμός αλυσίδων)

και

χρόνος υπολογισμού αλυσίδας = (κύκλοι μιας επανάληψης) X (μήκος αλυσίδας)

Βλέπουμε από τα παραπάνω αποτελέσματα ότι για να παραχθούν πίνακες ουράνιου τόξου με σχετικά καλή πιθανότητα επιτυχίας εύρεσης κλειδιού, χρειάζονται μέχρι και μέρες. Ο κυριότερος λόγος είναι το μεγάλο μήκος αλυσίδας που πρέπει να υπάρχει ώστε να αυξηθεί και η πιθανότητα επιτυχίας.

Για να συγκρίνουμε τα αποτελέσματα αυτά με τους αντίστοιχους χρόνους δημιουργίας πινάκων σε software, χρησιμοποιήσαμε το rainbow crack project [22]. Η συγκεκριμένη ομάδα παρέχει μια software υλοποίηση της παραγωγής των πινάκων ουράνιου τόξου για τους κυριότερους κρυπτογραφικούς αλγόριθμους (π.χ. MD5, LMHash, NTLM). Δυστυχώς, ο

αλγόριθμος A5/3 δεν υποστηριζόταν αλλά δινόταν η δυνατότητα (από την έκδοση 1.3 και μετά του rainbow crack project) να ενσωματώσουμε τον δικό μας αλγόριθμο στο συγκεκριμένο project και να μπορέσουμε να δημιουργήσουμε τους πίνακές μας.

Κάνοντας compile έναν κώδικα C από το συγκεκριμένο project, δημιουργούμε ένα dynamic link library αρχείο (.dll). Το αρχείο αυτό, το χρησιμοποιούμε σαν παράμετρο για την συνάρτηση rtgen(). Η συνάρτηση rtgen είναι αυτή που δημιουργεί τους πίνακες ουράνιου τόξου. Δεν μπορούσαμε να πάρουμε αποτελέσματα για μεγάλες τιμές μήκους αλυσίδας καθώς θα έπρεπε να τρέχει ο συγκεκριμένος κώδικας για εβδομάδες. Το μέγιστο μήκος αλυσίδας που χρησιμοποιήσαμε ήταν ενδεικτικά 100,000 , σε πραγματικούς πίνακες ουράνιου τόξου η τιμή αυτή μπορεί να είναι πολλαπλάσια. Η rtgen() μας δίνει επίσης πληροφορίες σχετικά με τους χρόνους παραγωγής των πινάκων. Να αναφέρουμε ότι ο επεξεργαστής που χρησιμοποιήθηκε για την υλοποίηση σε software ήταν ένας Intel dual-Core i5 @ 2.26 GHz, με προφανώς πολλές φορές πολλαπλάσιο ρολόι σε σχέση με το δικό μας (91.896 MHz).

Παραθέτουμε τα αποτελέσματα που πήραμε για τους χρόνους παραγωγής των πινάκων με την software προσέγγιση. Οι μετρήσεις πάρθηκαν για ίδιο βάθος μνήμης με την μέγιστη που μπορούμε να έχουμε και σε hardware έτσι ώστε να είναι συγκρίσιμα τα μεγέθη. Οι πίνακες που δημιουργούνται από την συνάρτηση rtgen() δεν είναι ταξινομημένοι. Προκειμένου να ταξινομηθούν, χρησιμοποιήσαμε την συνάρτηση rtsort() που επίσης παρέχεται από το rainbowcrack project. Η συνάρτηση αυτή όπως είναι λογικό, χρειάζεται κάποιον επιπλέον χρόνο προκειμένου να ταξινομήσει τους πίνακες που παράγουμε, οι χρόνοι αυτοί φαίνονται στον παρακάτω πίνακα.

Μήκος αλυσίδας	Χρόνος παραγωγής πινάκων (sec)	Χρόνος ταξινόμησης πινάκων (sec)	Συνολικός χρόνος (sec)
1	42.77	9.14	51.91
10,000	489.76	92.18	581.94
20,000	918.29	154.25	1,072.54
100,000	4,415.43	548.07	4,963.35

Πίνακας 7.4 Χρόνοι παραγωγής πινάκων για software υλοποίηση

Συγκρίνουμε τους παραπάνω συνολικούς χρόνους με την αντίστοιχη υλοποίηση σε hardware που έχουμε κάνει, χωρίς τον παραλληλισμό. Θυμίζουμε ότι στην hardware υλοποίηση έχουμε συμπεριλάβει και την διαδικασία ταξινόμησης των πινάκων (μέσω της hash function που δημιουργήσαμε).

Μήκος αλυσίδας, t	Χρόνος υπολογισμού πίνακα, sec (hardware)	Χρόνος υπολογισμού πίνακα, sec (software)	speedup
1,000	5.705	51.91	9.1
10,000	57.05	581.94	10.2
20,000	114.1	1,072.54	9.4
100,000	570.5	4,963.35	8.7

Πίνακας 7.5 Σύγκριση με software υλοποίηση για 1 μηχάνημα

Το speedup είναι αρκετά ικανοποιητικό καθώς χρησιμοποιούμε μία μόνο σχεδίαση. Δεν έχουμε αποτελέσματα για πίνακες με μεγαλύτερο μήκος αλυσίδας καθώς θα χρειαζόμασταν να τρέχει για αρκετές μέρες ή και εβδομάδες η software υλοποίηση.

Εφ όσον η συνολική σχεδίασή μας ήταν πολύ μικρή, (1% της διαθέσιμης χωρητικότητας στην FPGA) προχωρήσαμε σε παραλληλισμό της. Θεωρούμε ότι η σχεδίασή μας είναι πλήρως παραλληλοποιήσιμη καθώς κάθε μηχάνημα θα παράγει διαφορετικές αλυσίδες. Για να γίνει αυτό, χρησιμοποιούμε διαφορετικές αρχικοποιημένες τιμές (seeds) στους counters που δίνουν τιμές στα starting points.

Συνολικά, μπορέσαμε να χρησιμοποιήσουμε το 81% της διαθέσιμης μνήμης στην FPGA. Για κάθε παραλληλισμό που πραγματοποιούσαμε (x2, x4, x8, x32, x64) και επειδή έχουμε συγκεκριμένους περιορισμούς στην BRAM που μπορούμε να χρησιμοποιήσουμε, αυξάναμε τον αριθμό των μνημών και ταυτόχρονα μειώναμε το μέγεθός τους. Ο αριθμός των εκάστοτε μνημών είναι προφανώς όσος και η παραλληλοποίηση και το μέγεθος των μνημών θα ισούται κάθε φορά με το αρχικό μέγεθος μνήμης προς τον παραλληλισμό που έχουμε. Μεγαλύτερο παραλληλισμό δεν μπορούσαμε να επιτύχουμε καθώς ξεπερνούσαμε τους διαθέσιμους πόρους της fpga.

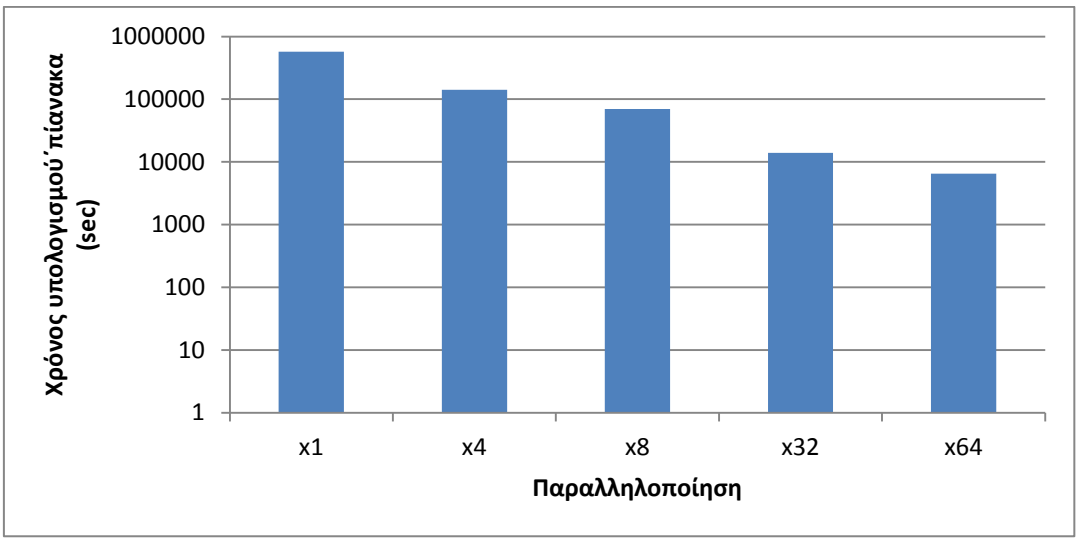
Τα αποτελέσματα του slice utilization για την παραλληλοποιημένη αρχιτεκτονική μας είναι

Παραλληλοποίηση	Slice utilization	Slice utilization (%)	Maximum frequency
x1	2195/207360	1%	91.896 MHz
x4	8853/207360	4.2%	91.783 MHz
x8	16381/207360	7.9%	91.790 MHz
x32	72576/207360	35%	90.985 MHz
x64	167961/207360	81%	90.763 MHz

Πίνακας 7.6 Αποτελέσματα synthesize για παράλληλα μηχανήματα

Παρατηρούμε ότι υπάρχει μια μικρή πτώση στην συχνότητα του ρολογιού που όμως δεν μας επηρεάζει ιδιαίτερα στα αποτελέσματά μας. Αρχικά, θα δούμε την βελτίωση που προκύπτει από τον παραλληλισμό σε hardware. Στην παρακάτω γραφική βλέπουμε την βελτίωση στον

χρόνο παραγωγής των πινάκων.



Πίνακας 7.7 Βελτίωση με παραλληλοποιημένη αρχιτεκτονική

Προκειμένου να συγκρίνουμε τις αρχιτεκτονικές αυτές με software, π.χ. στην περίπτωση των 64 παράλληλων σχεδιάσεων, θα συγκρίνουμε τον χρόνο που κάνει για να παράγει 1 πίνακα σε software μια μηχανή βάθους 1 64 του συνολικού βάθους που παράγει το hardware.

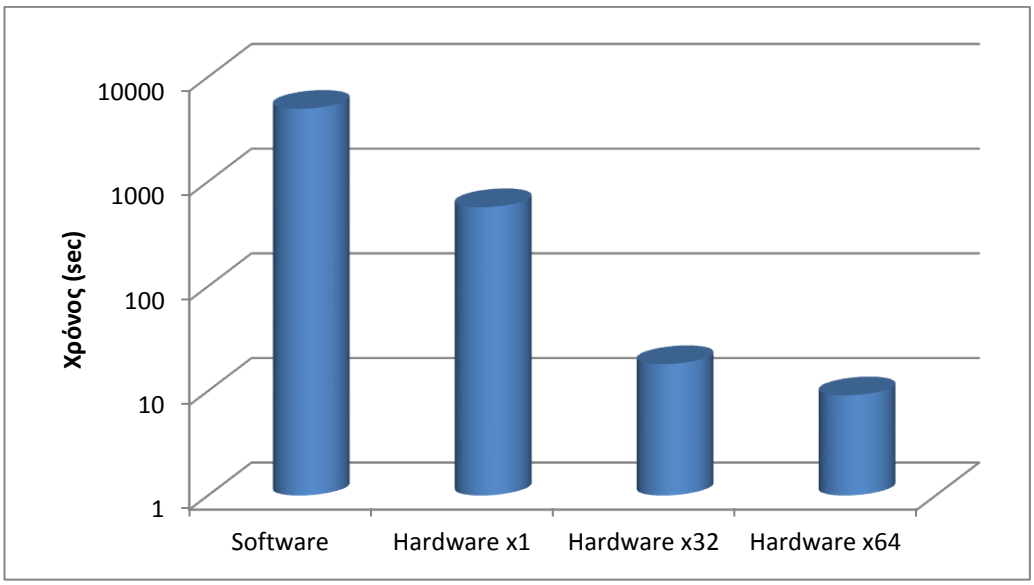
Έτσι, δημιουργήσαμε τον παρακάτω πίνακα που δείχνει τους χρόνους παραγωγής των πινάκων στις x32 και x64 αρχιτεκτονικές.

Παραλληλοποίηση	Αριθμός αλυσίδων για κάθε πίνακα, m	Μήκος αλυσίδα, t	Clock cycle (ns)	Χρόνος υπολογισμού αλυσίδας (ns)	Χρόνος υπολογισμού πίνακα (ns)	Χρόνος υπολογισμού πίνακα (sec)
x32	65536	100,000	11.035	8,828,000	18,079,744,000	18,079
x64	65536	100,000	11.104	8,883,200	9,096,396,800	9,09

Πίνακας 7.8 Χρόνοι για παραλληλοποιημένες αρχιτεκτονικές

Στον παραπάνω πίνακα η πιθανότητα επιτυχίας είναι σχετικά μικρή παρ όλα αυτά, ο σκοπός είναι να συγκρίνουμε τον συνολικό χρόνο παραγωγής των πινάκων με την αντίστοιχη υλοποίηση σε software.

Συγκριτικά με την υλοποίηση σε software λοιπόν, έχουμε το παρακάτω αποτέλεσμα



Πίνακας 7.9 Σύγκριση software-hardware υλοποίησης

Είναι εμφανές ότι επιτυγχάνεται μεγάλη βελτίωση στον χρόνο παραγωγής των πινάκων, της τάξης του x540 στην καλύτερη περίπτωση. Αν μπορούσαμε να έχουμε μεγαλύτερο βάθος μνήμης, άρα και περισσότερες εγγραφές στον πίνακά μας θα είχαμε καλύτερα αποτελέσματα και για την πιθανότητα επιτυχίας των πινάκων.

7.2 Μελλοντική δουλειά- βελτιώσεις

Σε σχέση με όσα παρουσιάσαμε, πολλές είναι οι βελτιώσεις που θα μπορούσαν να επιτευχθούν, με κύριο στόχο την βελτίωση της ταχύτητας στην παραγωγή των πινάκων μας.

Χρησιμοποιώντας διαφορετικές αρχιτεκτονικές για την υλοποίηση του αλγορίθμου κρυπτογράφησης, θα μπορούσαμε να έχουμε καλύτερα αποτελέσματα. Υπάρχουν αρκετές διαφορετικές υλοποιήσεις του συγκεκριμένου αλγορίθμου με κυριότερες τις [10] και [13]. Οι συγκεκριμένες υλοποιήσεις παρουσιάζουν μεγάλες βελτιστοποιήσεις στην αποδοτικότητα του αλγορίθμου και θα μπορούσαν να βελτιώσουν αρκετά τον χρόνο δημιουργίας των πινάκων.

Επίσης, στην αρχιτεκτονική που έχουμε φτιάξει για την παραγωγή των πινάκων ουσιαστικά δεν εκμεταλλευόμαστε το γεγονός της pipelined αρχιτεκτονικής του αλγορίθμου μας. Δηλαδή, αν και η είσοδος του κρυπτογραφικού αλγορίθμου μας μπορεί να δεχτεί διαφορετικό σήμα κάθε χρονική στιγμή, εμείς εισάγουμε καινούρια τιμή κάθε 8 κύκλους. Για να μετατρέψουμε όλη την αρχιτεκτονική της δημιουργίας των πινάκων ουράνιου τόξου σε pipelined θα έπρεπε να κάνουμε αρκετές αλλαγές στην ήδη υπάρχουσα αρχιτεκτονική οι οποίες θα έχουν και επιπτώσεις στο συνολικό μέγεθος της σχεδίασής μας. Συνεπώς, αφήνεται η συγκεκριμένη εργασία για μελλοντική δουλειά.

Η υλοποίησή μας έγινε για μια Virtex5 XC5VLX330T η οποία είχε σαν μειονέκτημα την μικρή διαθέσιμη μνήμη. Σαν μελλοντική δουλειά θα μπορούσαμε να μεταφέρουμε απευθείας τα rainbow chains που δημιουργούνται στην SRAM και από εκεί να τα μεταφέρουμε σε έναν σκληρό δίσκο. Η sram που χρησιμοποιούμε επίσης είναι μικρή σε χωρητικότητα οπότε θα μπορούσε να αντικατασταθεί από μια πιο γρήγορη και πιο μεγάλη DDR μνήμη.

Επίσης, η όλη σχεδίαση μπορεί να κατέβει στην συγκεκριμένη fpga και να μελετηθεί περαιτέρω, σε μια μεγαλύτερη σχεδίαση. Για παράδειγμα, θα μπορούσε να υλοποιηθεί και το on-line κομμάτι της αποκρυπτογράφησης, δηλαδή η διαδικασία εύρεσης του κλειδιού μέσα στους πίνακες ουράνιου τόξου. Με βάση την δουλειά που έχει γίνει στην διαδικασία παραγωγής των πινάκων κάτι τέτοιο θα ήταν εφικτό.

Βιβλιογραφία

1. **P. Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off**. Proc. of Advances in Cryptology, 23th Annual International Cryptology Conference, Santa Barbara, CA. USA, 2003.

2. **3rd Generation Partnership Project**, Technical Specification Group Services and System Aspects, 3G Security, Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 1: KASUMI Specification, V3.1.1, 2001

3. **3rd Generation Partnership Project**, Technical Specification Group Services and System Aspects, 3G Security, Specification of the A5/3 Encryption Algorithms for GSM and ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 4: Design and evaluation report, V6.1.0, 2002.

4. **3rd Generation Partnership Project**, Technical Specification Group Services and System Aspects, 3G Security, Specification of the A5/3 Encryption Algorithms for GSM and ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 2 : Implementor 's test data V6. 1.0 (2002-12)

5. **Kostas Theocharoulis, Ioannis Papaefstathiou, Charalampos Manifavas**: Implementing Rainbow Tables in High-End FPGAs for Super-Fast Password Cracking. pp.287-290, 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines, 2009

6. A5/1 Security Project, Creating A5/1 Rainbow Tables, 2009. Available online at http://reflextor.com/trac/a51

7. **Martin E. Hellman** : A Cryptanalytic Time-Memory Trade-off. IEEE Transactions on Information Theory, IT-26:401–406, 1980.

8. **Alex Biryukov** : Block Ciphers and Stream Ciphers: The State of the Art. Cryptology ePrint Archive, 2004, Report 2004/094, available on-line at http://eprint.iacr.org/2004/094.

9. **Orr Dunkelman, Nathan Keller, Adi Shamir** : A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony. International Crytology Conference - CRYPTO , pp. 393-410, 2010

10. **Tomas Balderas-Contreras, Rene A. Cumplido Parra** : An Efficient Hardware Implementation of the KASUMI Block Cipher for Third Generation Cellular Networks. In: Proceedings of the global signal processing conference, GSPx 2004, Santa Clara, CA; 2004.

11. **Eli Biham, Orr Dunkelman, Nathan Keller** : A Related-Key Rectangle Attack on the Full KASUMI. Advances in Cryptology - Asiacrypt '05, Lecture Notes in Computer Science, Vol. 3788, pp. 443–461, Springer-Verlag, 2005

12. **N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede**, "Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password Cracking," ARC 2006, Delft, The Netherlands, March 1-3, 2006

13. **Paris Kitsos, Michalis D. Galanis, and Odysseas G. Koufopavlou :** High speed hardware implementations of the KASUMI block cipher. In: Proceedings of the 2004 IEEE international symposium on circuit and systems ISCAS'04, Vancouver, Canada; 2004.

14. **ETSI/SAGE**. f8 and f9 specification. Specification of the 3GPP Confidentiality and Integrity Algorithms Document 1, ETSI/SAGE, September 2000. Available from http://www.etsi.org/dvbandca/3GPP/3gppspecs.htm.

15. **J. Quisquater, F.-X. Standaert, G. Rouvroy, and J.D. Legat**, "A cryptanalytic time-memory trade-off: First FPGA implementation," FPL 1998, Tallinn, Estonia, Au31 - Sep 3, 1998.

16. **Christos Xenakis and Lazaros Merakos** : Security in 3rd Generation Mobile Networks. Computer Communications-COMCOM, Elsevier Science, Vol. 27, No. 7, May 2004, pp. 638-650.

17. Cypress, *CY7C1041 256K x 16 Static RAM* Document #: 001-06496 Rev. *A

Βιβλία

18. Handbook of applied cryptography Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone

Ιστοσελίδες

19. http://xilinx.com

20. http://en.wikipedia.org

21. http://www.cypress.com

22. http://project-rainbowcrack.com/ (Project RainbowCrack)

23. http://lists.lists.reflextor.com/pipermail/a51/

24. http://www.cryptogram.org/