ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση σε αναδιατασσόμενη λογική των πινάκων ουρανίου τόξου (rainbow tables) για την αποκρυπτογράφηση των κρυπταλγόριθμων Lmhash, MD5, SHA1

ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ: 2007 – 2008

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ ΦΟΙΤΗΤΗ
ΘΕΟΧΑΡΟΥΛΗ ΚΩΝΣΤΑΝΤΙΝΟΥ
ΑΜ: 2001030072

ΕΠΙΒΛΕΠΩΝ :
ΠΑΠΑΕΥΣΤΑΘΙΟΥ ΙΩΑΝΝΗΣ
Επίκουρος Καθηγητής Πολυτεχνείου Κρήτης

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ Κ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση σε αναδιατασσόμενη λογική των πινάκων ουρανίου τόξου (rainbow tables) για την αποκρυπτογράφηση των κρυπταλγόριθμων Lmhash, MD5, SHA1

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΕΟΧΑΡΟΥΛΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ
ΑΜ: 2001030072

Επιτροπή

Επίκουρος Καθηγητής Πολυτεχνείου Κρήτης	**κ. ΠΑΠΑΕΥΣΤΑΘΙΟΥ ΙΩΑΝΝΗΣ**
Καθηγητής Πολυτεχνείου Κρήτης	**κ. ΔΟΛΛΑΣ ΑΠΟΣΤΟΛΟΣ**
Αναπληρωτής Καθηγητής Πολυτεχνείου Κρήτης	**κ. ΠΝΕΥΜΑΤΙΚΑΤΟΣ ΔΙΟΝΥΣΗΣ**

.

ΠΡΟΛΟΓΟΣ

Η παρούσα Διπλωματική Εργασία με τίτλο «Υλοποίηση σε αναδιατασσόμενη λογική των πινάκων ουρανίου τόξου (rainbow tables) για την αποκρυπτογράφηση των κρυπταλγόριθμων Lmhash, MD5, SHA1» αφορά τη υλοποίηση των rainbow tables σε VHDL για τους κρυπταλγόριθμους LM hash – MD5- SHA1, με τη χρήση FPGA .

Η Διπλωματική Εργασία ανατέθηκε στον τελειόφοιτο Κων/νο Θεοχαρούλη από τον Επίκουρο Καθηγητή του Τμήματος Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών του Πολυτεχνείου Κρήτης, Ιωάννη Παπαευσταθίου μετά από συνεννόηση με τους καθηγητές κ. Απόστολο Δόλλα και κ. Διονυση Πνευματικάτο καθώς και με τον Δρ. κ. Χαράλαμπο Μανιφάβα .

Για την εκπόνηση αυτής της εργασίας θα ήθελα να ευχαριστήσω θερμά τον Καθηγητή μου Ιωάννη Παπαευσταθίου για την συνεχή του καθοδήγηση. Το ενδιαφέρον του, όχι μόνο για την εκπόνηση αυτής της εργασίας, αλλά και για την εις βάθος κατανόηση της ουσίας του προβλήματος.

Ευχαριστίες επίσης, θα ήθελα να εκφράσω, στο Μηχανικό Μειντάνη Δημήτρη και στον Δρ. Philippe Oechslin που με ενδιαφέρον απάντησαν σε όλες τις ερωτήσεις μου, και με βοήθησαν στις δυσκολίες της διπλωματικής μου εργασίας, καθώς και στους υπόλοιπους Μηχανικούς του *Εργαστηρίου Μικροεπεξεργαστών και Υλικού* για τις όποιες βοήθειες που έδωσαν το διάστημα αυτό.

Τέλος, επειδή με την εργασία αυτή, ολοκληρώνονται και οι σπουδές μου ως προπτυχιακού φοιτητή θα ήθελα να ευχαριστήσω την οικογένειά μου, που με υποστήριξε σε όλες μου τις αποφάσεις με κάθε τρόπο.

Χανιά , Απρίλης 2008

ΠΕΡΙΛΗΨΗ

Η κρυπτογραφία συμμετρικού κλειδιού ασχολείται με αλγόριθμους που χρησιμοποιούν ένα κρυφό κλειδί για να προσφέρουν εμπιστευτικότητα, προσδιορισμούς και επικύρωση δεδομένων.

Ένα βασικό πρόβλημα στη κρυπτογραφία συμμετρικού κλειδιού είναι ο προϋπολογισμός των συνόψεων ή η αναστροφή της μονόδρομης συνάρτησης. Για παράδειγμα μια επίθεση ωμής βίας σε να κρυπτογραφημένο block γνωστής εισόδου χρησιμοποιεί την αντιστοίχηση κλειδιού στο κρυπτογραφημένο block, το οποίο πρέπει να είναι μια μονόδρομη συνάρτηση.

Αν δεν είναι γνωστή κάποια μέθοδος παράκαμψης και η πράξη έχει n-bit αποτέλεσμα τότε υπάρχουν 2 μέθοδοι. Η πρώτη μπορεί να διενεργήσει μια ενδελεχής έρευνα πάνω σε ένα μέσο όρο από 2^{n-1} τιμές μέχρι να φτάσει στο αποτέλεσμα. Η δεύτερη λύση είναι να προϋπολογίσει και να αποθηκεύσει 2^n ζευγάρια τιμών εισόδου – εξόδου σε ένα πίνακα(για μια τυχαία πράξη αυτό δε θα έχει αποτέλεσμα να βρούμε διαφορετικές τιμές – εάν το εύρος εισόδου είναι αρκετά μεγάλο , διότι πρέπει να ερευνηθούν $n * 2^n$ στοιχεία)

Η επίθεση με ανταλλαγή χρόνου – μνήμης δημιουργήθηκε από τον Hellman το 1980 [***βιβλ.12***] και προτείνει μια λύση που βρίσκεται μεταξύ των 2 λύσεων. Ο χρόνος προϋπολογισμού παραμένει της τάξης του 2^n άλλη η πολυπλοκότητα της μνήμης είναι $2^{n/3}$ και η αναστροφή μιας μοναδικής τιμής απαιτεί μόνο $2^{n/3}$ υπολογισμούς πράξεων. Ο Fiat κ Naor [***βιβλ. 25***] προτείνουν μια πιο γενική και αυθεντική μεταβλητή με το κόστος του επιπλέον φόρτου εργασίας και μνήμης. Ο Kusuda και Matsumoto [***βιβλ. 16***] γενίκευσαν την μέθοδο Hellman αυτοί παράγουν αυστηρότερα όρια στη πιθανότητα επιτυχίας και δίνουν σχέσεις μεταξύ της πολυπλοκότητας της μνήμης , της πολυπλοκότητα της επεξεργασίας και την πιθανότητα επιτυχίας. Σημειώνουμε ότι για πιο πολύπλοκα και σύνθετα προβλήματα ανταλλαγής χρόνου / μνήμης / δεδομένων κρυπτανάλυσης αναλυτική περιγραφή γίνεται από τους Biryukov και Shamir[***βιβλ. 2***]

Η βασική ιδέα του Hellman βελτιώθηκε το 1982 από τον Rivest που πρότεινε να χρησιμοποιούνται διακριτά σημεία για να μειωθεί ο χρόνος των προσβάσεων στη μνήμη. Αυτή η ιδέα διαμορφώθηκε από τον Borst [***βιβλ. 26***] και τον Stern [***βιβλ. 27***]. Το πρώτο σχέδιο σε FPGA αυτής της μεθόδου προτάθηκε από τον Quisquater για ένα 40 bits DES , παρουσίασαν επίσης την εκτίμηση κόστους για την κρυπτανάλυση ενός πλήρους DES (56 bits) [***βιβλ. 19***]. Μια πιο γενική ανάλυση πλήρους κόστους της ανταλλαγής χρόνου –μνήμης και δίχως διακριτά σημεία έχει παρασχεθεί από το Wiener [***βιβλ. 28***]

Στο Crypto 2003 [***βιβλ. 3***] , ο Oechslin πρότεινε τη χρήση των λεγόμενων πινάκων ουρανίου τόξου, για τους προϋπολογισμούς αυτή η μέθοδο συνδυάζει τα πλεονεκτήματα της προσέγγισης διακριτών σημείων (δηλαδή μειωμένος αριθμός προσβάσεων στη μνήμη) με την υψηλή πιθανότητα επιτυχίας και την ευκολότερη ανάλυση της αρχικής μεθόδου του Hellman. Ανέπτυξε περισσότερες λεπτομέρειες στη [***βιβλ. 18***]

Σε αυτή την εργασία προτείνουμε μια πλατφόρμα FPGA για την δημιουργία των rainbow tables για την κρυπτανάλυση των κρυπταλγόριθμων Lmhash που εντάσσεται στη συμμετρική κρυπτογραφία και μας απασχόλησε περισσότερο καθώς και για τους MD5 και SHA-1 στους οποίους δεν υπάρχει κλειδί αλλά ανήκουν στα δίκτυα αντικατάστασης. Ο στόχος της εφαρμογής είναι μια υπάρχων FPGA πλατφόρμα VIRTEX 5 της XILINX στην οποία δουλέψαμε μόνο με simulate ενώ παράλληλα εργασθήκαμε με στη VIRTEX2PRO την οποία και διαθέταμε στο εργαστήριο.

Η εργασία είναι οργανωμένη ως ακόλουθα :

Αρχικά κάνουμε μια αναφορά στην κρυπτογραφία και στους αλγόριθμους που υλοποιήσαμε και γενικά στοιχεία για τους πίνακες ουρανίου τόξου. Περιγράφουμε το θεωρητικό υπόβαθρο καθώς και μερικούς ορισμούς όπως και διευκρινίσεις σχετικά με τη περίπτωση μας . Στο επόμενο τμήμα παρουσιάζονται οι λεπτομέρειες της υλοποίησης σε FPGA καθώς και οι βελτιστοποιήσεις που κάναμε .

Τέλος παρουσιάζονται τα συμπεράσματα καθώς και μελλοντικές εργασίες που θα μπορούσαν να γίνουν και η βιβλιογραφία στο τελευταίο τμήμα.

ABSTRACT

This paper presents a hardware architecture for passwords MD5, LM-hash and SHA1 cracking using Hellman's time-memory trade-off.

It is a hardware design for a key search machine based on the the rainbow variant proposed by Oechslin.

We estimate that an FPGA implementation of the function that crreates the rainbow tables can run faster on a Virtex5 than on software

Our design targets passwords of length 56bits. We estimate that recovering an individual password requires a few minutes.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΥΡΙΟ ΚΕΙΜΕΝΟ

1. ΕΙΣΑΓΩΓΗ

1.1. Γενικά για την κρυπτογραφία

Η λέξη **κρυπτογραφία** προέρχεται από τα συνθετικά "κρυπτός" + "γράφω" και είναι ένας επιστημονικός κλάδος που ασχολείται με την μελέτη, την ανάπτυξη και την χρήση τεχνικών κρυπτογράφησης και αποκρυπτογράφησης με σκοπό την απόκρυψη του περιεχομένου των μηνυμάτων.

Η κρυπτογραφία είναι ένας κλάδος της επιστήμης της κρυπτολογίας, η οποία ασχολείται με την μελέτη της ασφαλούς επικοινωνίας. Ο κύριος στόχος της είναι να παρέχει ασφαλή επικοινωνία και χωρίζεται σε δύο κλάδους: την **Κρυπτογραφία** και την Κρυπτανάλυση.

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

Η κρυπτογραφία παρέχει 4 βασικές λειτουργίες :

- ***Εμπιστευτικότητα***: Η πληροφορία προς μετάδοση είναι προσβάσιμη μόνο στα εξουσιοδοτημένα μέλη. Η πληροφορία είναι ακατανόητη σε κάποιον τρίτο.
- ***Ακεραιότητα***: Η πληροφορία μπορεί να αλλοιωθεί μόνο από τα εξουσιοδοτημένα μέλη και δεν μπορεί να αλλοιώνεται χωρίς την ανίχνευση της αλλοίωσης.
- ***Μη απάρνηση***: Ο αποστολέας ή ο παραλήπτης της πληροφορίας δεν μπορεί να αρνηθεί την αυθεντικότητα της μετάδοσης ή της δημιουργίας της.

•***Πιστοποίηση***: Οι αποστολέας και παραλήπτης μπορούν να εξακριβώνουν τις ταυτότητές τους καθώς και την πηγή και τον προορισμό της πληροφορίας με διαβεβαίωση ότι οι ταυτότητές τους δεν είναι πλαστές.

1.2. Ορολογία

Κρυπτογράφηση (*encryption*) ονομάζεται η διαδικασία μετασχηματισμού ενός μηνύματος σε μία ακατανόητη μορφή με την χρήση κάποιου κρυπτογραφικού αλγορίθμου ούτως ώστε να μην μπορεί να διαβαστεί από κανέναν εκτός του νόμιμου παραλήπτη.

Η αντίστροφη διαδικασία όπου από το κρυπτογραφημένο κείμενο παράγεται το αρχικό μήνυμα ονομάζεται **αποκρυπτογράφηση (*decryption*)**.

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

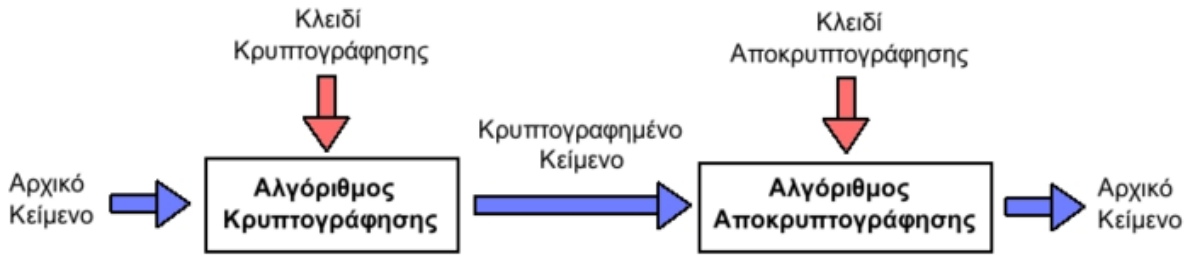
Αρχικό κείμενο (*plaintext*) είναι το μήνυμα το οποίο αποτελεί την είσοδο σε μία διεργασία κρυπτογράφησης.

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

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

Κρυπτανάλυση (*cryptanalysis*) είναι μία επιστήμη που ασχολείται με το "σπάσιμο" κάποιας κρυπτογραφικής τεχνικής ούτως ώστε χωρίς να είναι γνωστό το κλειδί της κρυπτογράφησης, το αρχικό κείμενο να μπορεί να αποκωδικοποιηθεί.

Η διαδικασία της κρυπτογράφησης και της αποκρυπτογράφησης φαίνεται στο παρακάτω σχήμα.



Σχήμα 1.1. Διαδικασία της κρυπτογράφησης και της αποκρυπτογράφησης

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

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

1.3. Ιστορικά

Η κρυπτογραφία χωρίζεται σε 3 περιόδους ιστορικά:

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

Κατά την διάρκεια αυτής της περιόδου αναπτύχθηκε μεγάλο πλήθος μεθόδων και αλγορίθμων κρυπτογράφησης, που βασιζόταν κυρίως σε απλές αντικαταστάσεις γραμμάτων. Όλες αυτές δεν απαιτούσαν εξειδικευμένες γνώσεις και πολύπλοκες

συσκευές αλλά στηριζόταν στην ευφυΐα και την ευρηματικότητα των δημιουργών τους. Όλα αυτά τα συστήματα έχουν στις μέρες μας κρυπταναλυθεί και έχει αποδειχθεί, ότι, εάν μας είναι γνωστό ένα μεγάλο κομμάτι του κρυπτογραφημένου μηνύματος, τότε το αρχικό κείμενο μπορεί σχετικά εύκολα να επανακτηθεί.

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

Καλύπτει τους δύο παγκόσμιους πολέμους, εξαιτίας των οποίων (λόγω της εξαιρετικά μεγάλης ανάγκης που υπήρξε για ασφάλεια κατά την μετάδοση ζωτικών πληροφοριών μεταξύ των στρατευμάτων των χωρών) αναπτύχθηκε η κρυπτογραφία τόσο όσο δεν είχε αναπτυχθεί τα προηγούμενα 3000 χρόνια. Τα κρυπτοσυστήματα αυτής της περιόδου αρχίζουν να γίνονται πολύπλοκα, και να αποτελούνται από μηχανικές και ηλεκτρομηχανικές κατασκευές, οι οποίες ονομάζονται «κρυπτομηχανές». Η κρυπτανάλυση τους, απαιτεί μεγάλο αριθμό προσωπικού, το οποίο εργαζόταν επί μεγάλο χρονικό διάστημα ενώ ταυτόχρονα γίνεται εξαιρετικά αισθητή η ανάγκη για μεγάλη υπολογιστική ισχύ. Παρά την πολυπλοκότητα που αποκτούν τα συστήματα κρυπτογράφησης κατά την διάρκεια αυτής της περιόδου η κρυπτανάλυση τους είναι συνήθως επιτυχημένη. Οι Γερμανοί έκαναν εκτενή χρήση (σε διάφορες παραλλαγές) ενός συστήματος γνωστού ως Enigma .

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

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

Η εποχή της σύγχρονης κρυπτογραφίας αρχίζει ουσιαστικά με τον Claude Shannon, δημοσίευσε το έγγραφο «Θεωρία επικοινωνίας των συστημάτων μυστικότητας» το βιβλίο «Μαθηματική Θεωρία της Επικοινωνίας» μαζί με τον Warren Weaver. Καθιέρωσε μια στερεά θεωρητική βάση για την κρυπτογραφία και την κρυπτανάλυση. Εκείνη την εποχή η κρυπτογραφία εξαφανίζεται και φυλάσσεται από τις μυστικές υπηρεσίες κυβερνητικών επικοινωνιών όπως η NSA.(National security Agency). Πολύ

λίγες εξελίξεις δημοσιοποιήθηκαν ξανά μέχρι τα μέσα της δεκαετίας του '70, όταν όλα άλλαξαν.

Στα μέσα της δεκαετίας του '70 έγιναν δύο σημαντικές δημόσιες πρόοδοι (δηλ. μη-μυστικές). Πρώτα ήταν η δημοσίευση του σχεδίου προτύπου κρυπτογράφησης DES (Data Encryption Standard)στον ομοσπονδιακό κατάλογο της Αμερικής . Ο DES ήταν ο πρώτος δημόσια προσιτός αλγόριθμος κρυπτογράφησης που εγκρίνεται από μια εθνική αντιπροσωπεία. Η απελευθέρωση της προδιαγραφής της , υποκίνησε μια έκρηξη δημόσιου και ακαδημαϊκού ενδιαφέροντος για τα συστήματα κρυπτογραφίας. Ο DES αντικαταστάθηκε επίσημα από τον AES το 2001 Ο DES και οι ασφαλέστερες παραλλαγές του όπως ο 3DES ή TDES χρησιμοποιούνται ακόμα σήμερα, ενσωματωμένος σε πολλά εθνικά και οργανωτικά πρότυπα. Εντούτοις, το βασικό μέγεθος των 56-bit έχει αποδειχθεί ότι είναι ανεπαρκές να αντισταθεί στις επιθέσεις ωμής βίας (μια τέτοια επίθεση πέτυχε να σπάσει τον DES σε 56 ώρες). Κατά συνέπεια, η χρήση απλής κρυπτογράφησης με τον DES είναι τώρα χωρίς την αμφιβολία επισφαλής για χρήση στα νέα σχέδια των κρυπτογραφικών συστημάτων και μηνύματα που προστατεύονται από τα παλαιότερα κρυπτογραφικά συστήματα που χρησιμοποιούν DES.

1.4. Εφαρμογές κρυπτογραφίας

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

1.Ασφάλεια συναλλαγών σε τράπεζες δίκτυα - ATM
2.Κινητή τηλεφωνία (ΤΕΤΡΑ-ΤΕΤΡΑΠΟΛ-GSM)
3.Σταθερή τηλεφωνία (cryptophones)
4.Διασφάλιση Εταιρικών πληροφοριών
5.Στρατιωτικά δίκτυα (Τακτικά συστήματα επικοινωνιών μάχης)

6.Διπλωματικά δίκτυα (Τηλεγραφήματα)

7.Ηλεκτρονικές επιχειρήσεις (πιστωτικές κάρτες, πληρωμές)

8.Ηλεκτρονική ψηφοφορία

9.Ηλεκτρονική δημοπρασία

10.Ηλεκτρονικό γραμματοκιβώτιο

11.Συστήματα συναγερμών

12.Συστήματα βιομετρικής αναγνώρισης

13.Έξυπνες κάρτες

14.Ιδιωτικά δίκτυα (VPN)

15.Word Wide Web

16.Δορυφορικές εφαρμογές (δορυφορική τηλεόραση)

17.Ασύρματα δίκτυα (Hipper LAN, Bluetooth)

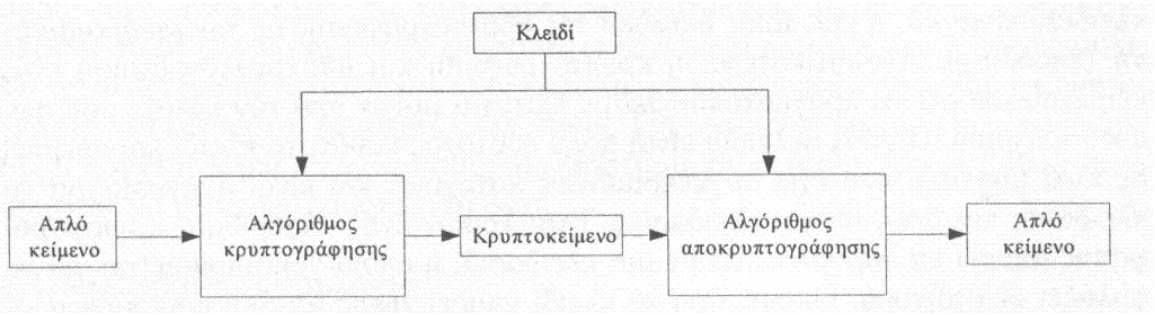
18.Συστήματα ιατρικών δεδομένων και άλλων βάσεων δεδομένων

19.Τηλεσυνδιάσκεψη - Τηλεφωνία μέσω διαδικτύου (VOIP)

2. Κρυπταλγόριθμοι που ασχοληθήκαμε

2.1 LM hash ή **LAN hash manager** είναι μία από τις διατάξεις που χρησιμοποιούν το Microsoft Lan Manager και τα Microsoft Windows , για να αποθηκεύσουν τους κωδικούς πρόσβασης των χρηστών, που είναι μικρότεροι από 15 χαρακτήρες σε μήκος. Αυτός ο τύπος hash είναι ο μόνος τύπος κρυπτογράφησης που χρησιμοποιείται στο Microsoft Lan Manager, ως εκ τούτου και το όνομα, καθώς και σε εκδόσεις των Microsoft Windows μέχρι και τα σημερινά Vista. [***βιβλ. 21***]

Πρόκειται για ένα συμμετρικό κρυπτοσύστημα δηλαδή το κλειδί της κρυπτογράφησης είναι το ίδιο με αυτό της αποκρυπτογράφησης. Τα συμμετρικά κρυπτοσυστήματα είναι και τα πιο διαδεδομένα χρονικά, καθότι τα ασύμμετρα κρυπτοσυστήματα εμφανίστηκαν επίσημα το 1976. Ένα συμμετρικό κρυπτοσύστημα αποτελείται από πέντε μέρη, όπως φαίνεται στο Σχήμα



Σχήμα 2.1 Συμμετρικό κρυπτοσύστημα

Το απλό κείμενο εισάγεται μαζί με το κλειδί στον αλγόριθμο κρυπτογράφησης. Όπως είναι φανερό στο παραπάνω σχήμα, το κλειδί είναι ανεξάρτητο του απλού κειμένου. Το αποτέλεσμα του αλγόριθμου κρυπτογράφησης είναι το κρυπτοκείμενο. Για δεδομένο απλό κείμενο, δύο διαφορετικά κλειδιά παράγουν δύο διαφορετικά κρυπτοκείμενα. Ο αλγόριθμος αποκρυπτογράφησης δέχεται ως είσοδο το κρυπτοκείμενο και το κλειδί το οποίο είναι το ίδιο με αυτό του αλγόριθμου κρυπτογράφησης. Ο αλγόριθμος αποκρυπτογράφησης εφαρμόζει τους αντίστροφους

μετασχηματισμούς από αυτούς του αλγόριθμου κρυπτογράφησης και επαναφέρει το κείμενο στην αρχική του μορφή, αυτήν του απλού κειμένου.

2.1.1 Περιγραφή του κρυπταλγόριθμου

Hash LM υπολογίζεται ως εξής.

1)Ο κωδικός πρόσβασης του χρήστη είναι μια συμβολοσειρά (OEM string -Original equipment manufacturer IBM) η οποία μετατρέπεται αρχικά σε κεφαλαία.

2)Αυτός ο κωδικός πρόσβασης είναι είτε null-padded (γεμισμένος μηδενικά έως το 14 χαρακτήρα αν το μήκος του είναι μικρότερο από 14) ή truncated (αποτελούμενος) από 14 bytes .

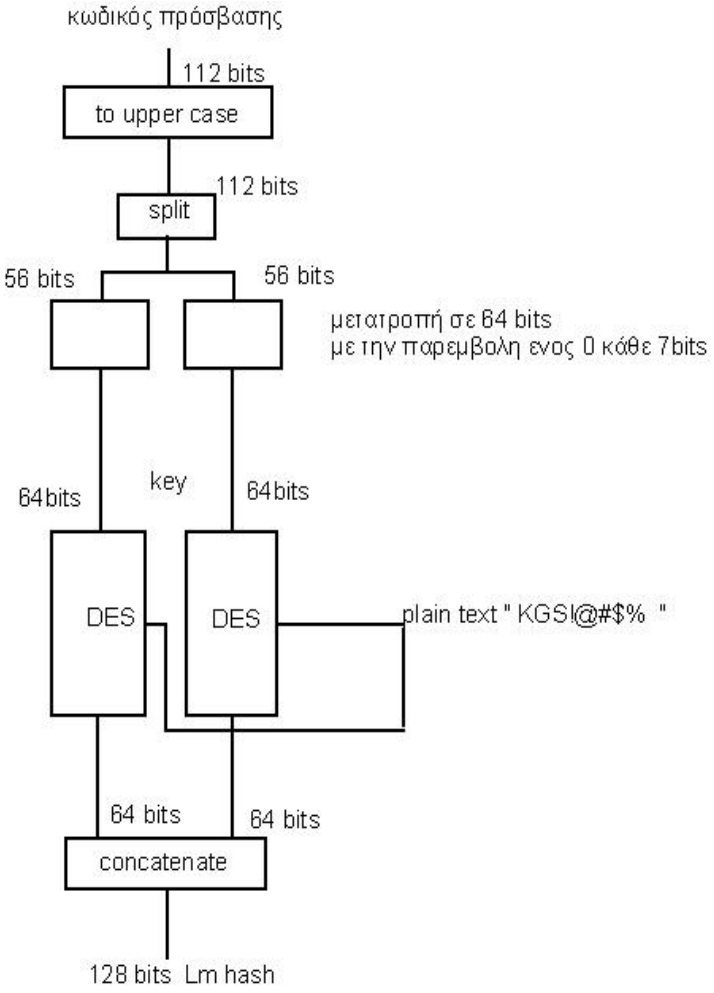
3)Ο "καθορισμένου μήκους" κωδικός πρόσβασης χωρίζεται σε δύο μισά 7- bytes το καθένα.

4)Αυτές οι τιμές χρησιμοποιούνται για να δημιουργήσουν δύο DES κλειδιά , με τη μετατροπή των 7 bytes σε 64 bits. Αυτό γίνεται με την παρεμβολή ενός μηδενικού bit μετά από κάθε επτά bits

5)Κάθε ένα από αυτά τα κλειδιά χρησιμοποιείται για μια DES-κρυπτογράφηση σε ένα συγκεκριμένο Plain text, τη σταθερή συμβολοσειρά ASCII "KGS!@#$%", με αποτέλεσμα δύο τιμές κρυπτογραφημάτων των 8-bytes η καθεμία.

6)Αυτές οι δύο τιμές κρυπτογραφημάτων συνδέονται για να διαμορφώσουν μια 16byte ποσότητα που αποτελεί την έξοδο του αλγορίθμου. Αυτό είναι και το LM Hash

Παρακάτω φαίνεται η δομή του Κρυπταλγόριθμου LM – hash .



Σχήμα 2.2 Διάγραμμα κρυπταλγόριθμου Lm hash

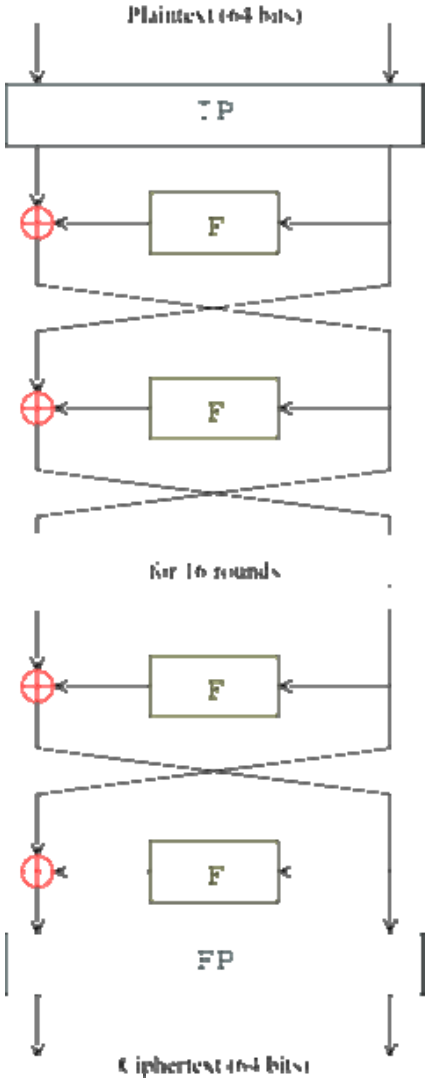
2.1.2. Αλγόριθμος DES

Κύριο στοιχείο του Lm hash είναι ο κρυπταλγόριθμος DES (**Data Encryption Standard**) ο οποίος είναι ο δημοφιλέστερος συμμετρικός-βασικός αλγόριθμος [***βιβλ. 29***] . Είναι ένας αλγόριθμος δέσμης τεμαχίζει σε τμήματα (blocks) το αρχικό κείμενο που πρόκειται να κρυπτογραφηθεί και κρυπτογραφεί κάθε τμήμα ξεχωριστά. που παίρνει μια καθορισμένου μήκους συμβολοσειρά plaintext και την μετασχηματίζει μέσω μιας σειράς περίπλοκων διαδικασιών σε ένα κρυπτογράφημα του ίδιου μήκους. Το μέγεθος της εισόδου είναι 64 bit.

Ο DES χρησιμοποιεί επίσης ένα κλειδί για να προσαρμόσει το μετασχηματισμό, έτσι ώστε η αποκρυπτογράφηση μπορεί μόνο να εκτελεσθεί από εκείνους που ξέρουν το ιδιαίτερο κλειδί που χρησιμοποιείται για τη κρυπτογράφηση. Το κλειδί αποτελείται φαινομενικά από 64 bits εντούτοις, μόνο 56 από αυτά αυτοί χρησιμοποιούνται πραγματικά από τον αλγόριθμο. Τα οκτώ bit χρησιμοποιούνται απλώς για τον έλεγχο της ισότητας, και απορρίπτονται έκτοτε.

2.1.2.1. Γενική δομή του DES

Η γενική δομή του αλγορίθμου παρουσιάζεται στο σχήμα



Σχήμα 2.3. γενική δομή Feistel DES

Υπάρχουν 16 ίδια στάδια της επεξεργασίας, μια αρχική και μια τελική αντιμετάθεση, που καλούνται *IP* και *FP*, οι οποίες είναι αντίστροφες (η IP "ανατρέπει" τη δράση του FP, και αντίστροφα). Η IP και το FP δεν έχουν σχεδόν καμία κρυπτογραφική σημασία.

Στην παραπάνω εικόνα μπορούμε να δούμε το βασικός μέρος του σχεδιασμού. Πριν από τους κύριους κύκλους, η είσοδος είναι διαιρεμένη σε δύο 32bit ποσότητες και επεξεργασμένες διαδοχικά σε αυτή τη σταυροειδής διάταξη που είναι γνωστή ως δίκτυο Feistel. Η δομή Feistel εξασφαλίζει ότι η αποκρυπτογράφηση και η κρυπτογράφηση είναι πολύ παρόμοιες διαδικασίες - η μόνη διαφορά είναι ότι τα subkeys – υποκλειδιά εφαρμόζονται στην αντίστροφη διάταξη κατά αποκρυπτογράφηση. Το υπόλοιπο του αλγορίθμου είναι ίδιο.

Το κόκκινο σύμβολο είναι η (XOR- exclusive OR) . Η *F* ανακατεύει το 1 από τα 2 block με μερικά κλειδιά. Το αποτέλεσμα από την λειτουργία αυτή συνδυάζεται έπειτα με το άλλο block, και ανταλλάσσουν θέσεις πριν από τον επόμενο κύκλο. Μετά από τον τελικό κύκλο, δεν ανταλλάσσονται αυτό είναι ένα χαρακτηριστικό γνώρισμα της δομής Feistel που κάνει την κρυπτογράφηση και την αποκρυπτογράφηση τις παρόμοιες διαδικασίες

2.1.2.2. Η συνάρτηση γύρου (F)

Η συνάρτηση γύρου f, που απεικονίζεται στο παραπάνω σχήμα , λειτουργεί στο μισό block(32 bit) σε έναν χρόνο και αποτελείται από τέσσερα στάδια:

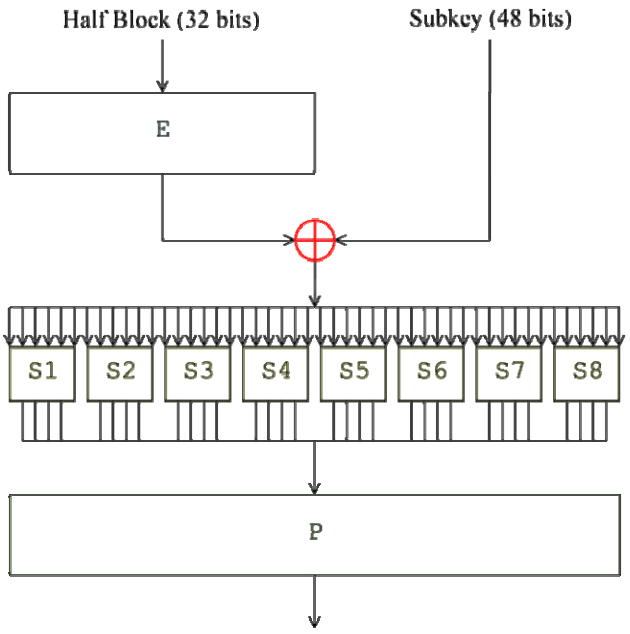
- *Επέκταση* – 32bit ποσότητα επεκτείνεται σε 48 bits χρησιμοποιώντας τη συνάρτηση μετάθεσης Ε όπως φαίνεται στο διάγραμμα, με την αντιγραφή κάποιων bits.

•*Μίξη με τα κλειδιά* - το αποτέλεσμα συνδυάζεται με ένα *subkey* χρησιμοποιώντας μια λειτουργία XOR. Δεκαέξι 48bit subkeys - ένα για κάθε κύκλο – προκύπτουν από το κύριο κλειδί χρησιμοποιώντας το πρόγραμμα κλειδιού.

•*Αντικατάσταση* - μετά από την ανάμιξη με το subkey, το block διαιρείται σε οκτώ κομμάτια των 6bits πριν περάσει στην επεξεργασία με τα S-boxes (ή αλλιώς κουτιά *αντικατάστασης*). Καθένα από τα οκτώ S-boxes αντικαθιστά έξι bits εισόδου με 4 bits εξόδου σύμφωνα με έναν μη γραμμικό μετασχηματισμό. Τα S-boxes παρέχουν τον πυρήνα της ασφάλειας DES . χωρίς αυτά, η κρυπτογραφική θα ήταν γραμμική , και κοινότοπα εύθραυστη.

•*Ανταλλαγή* - τελικά, τα 32 αποτελέσματα από s-boxes ρυθμίζονται εκ νέου σύμφωνα με μια σταθερή ανταλλαγή (P-box).

Στο παρακάτω σχήμα φαίνεται η συνάρτηση γύρου του Κρυπταλγόριθμου DES



Σχήμα 2.4. Η συνάρτηση γύρου

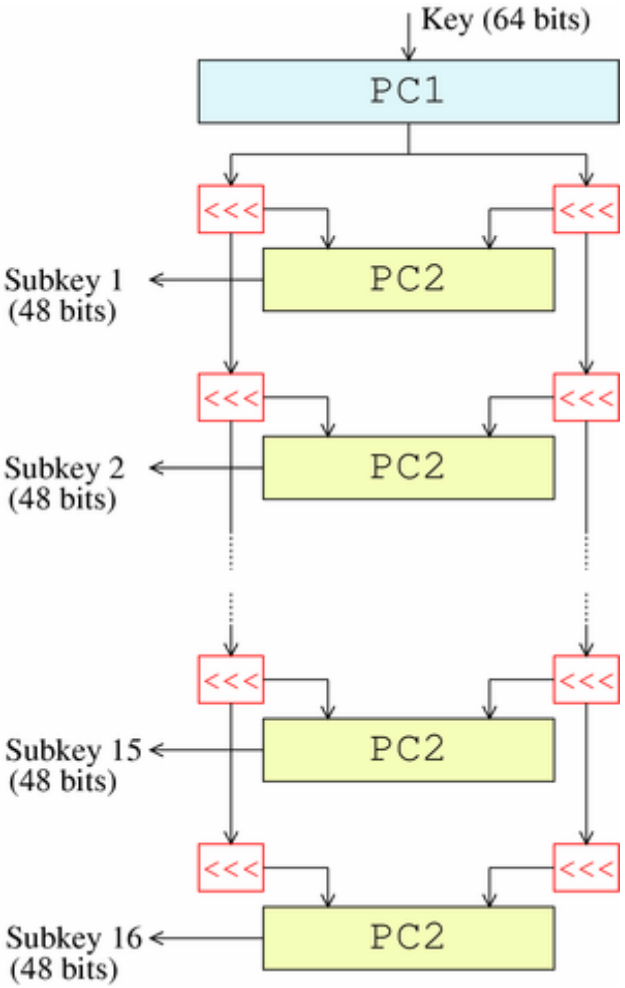
2.1.2.3. Το πρόγραμμα κλειδιού

Το παρακάτω σχήμα επεξηγεί το *βασικό πρόγραμμα* για την κρυπτογράφηση δηλαδή τον αλγόριθμο που παράγει τα subkeys. Αρχικά, τα 56 bits του κλειδιού επιλέγονται από τα αρχικά 64 από *τη συνάρτηση μετάθεσης επιλογής 1* (*PC1*) - τα υπόλοιπα 8 bit είτε απορρίπτονται είτε χρησιμοποιούνται ως κομμάτια ελέγχου ισότητας.

Τα 56 bits διαιρούνται σε 2 λέξεις των 28-bits και τροφοδοτούνται σε καταχωρητές ολίσθησης κάθε μισό αντιμετωπίζεται έκτοτε χωριστά. Στους διαδοχικούς κύκλους, και τα δύο μισά ολισθαίνουν προς τα αριστερά, και έπειτα 48 bits επιλέγονται από *τη συνάρτηση μετάθεσης επιλογής 2* (*PC2*) - 24 bits από το αριστερό μισό, και 24 από το δεξί. Οι περιστροφές (που δείχνονται "<<" στο διάγραμμα) σημαίνουν ότι ένα διαφορετικό σύνολο κομματιών χρησιμοποιείται σε κάθε subkey , κάθε κομμάτι χρησιμοποιείται σε περίπου 14 από τα 16 subkeys.

Το βασικό πρόγραμμα για την αποκρυπτογράφηση είναι παρόμοιο - πρέπει να παραγάγει τα κλειδιά στην αντίστροφη φορά. Ως εκ τούτου οι ολισθήσεις γίνονται προς τα δεξιά .

Το σχήμα μας δείχνει αναλυτικά το βασικό πρόγραμμα κλειδιού για την κρυπτογράφηση



Σχήμα 2.5. Το πρόγραμμα κλειδιού

2.1.3. Αδυναμίες Ασφάλειας Lm Hash

Αν και είναι βασισμένος στον DES , ο Lm hash μπορεί εύκολα να αποκωδικοποιηθεί λόγω **δύο αδυναμιών** στην εφαρμογή του.

Κατ' αρχάς, οι κωδικοί πρόσβασης πιο μεγάλοι από 7 χαρακτήρες διαιρούνται σε δύο κομμάτια και κάθε κομμάτι κωδικοποιείται χωριστά. Δεύτερον, όλα τα μικρά γράμματα στον κωδικό πρόσβασης μετατρέπονται σε κεφαλαία προτού ο κωδικός πρόσβασης, διατεθεί σε κομμάτια .

Η πρώτη αδυναμία είναι ότι μπορούμε να επιτεθούμε χωριστά σε κάθε μισό του κωδικού πρόσβασης. Ενώ υπάρχουν 2^{84} διαφορετικοί συνδυασμοί κωδικών πρόσβασης στους 14 χαρακτήρες γραμμάτων και ψηφίων, εντούτοις μόνο 2^{42} λόγω

του σπασίματος σε 2 κομμάτια χρησιμοποιώντας το ίδιο σύνολο χαρακτήρων. Περιορίζοντας τους χαρακτήρες σε κεφαλαία γράμματα και ψηφία μειώνουν περαιτέρω τον αριθμό δυνατοτήτων για κάθε μισό σε 2^{36}.

Με **επιθέσεις ωμής βίας** σε κάθε μισό χωριστά, οι σύγχρονοι υπολογιστές μπορούν να σπάσουν αλφα-νουμερικά κρυπτογραφημένα κείμενα (alphanumeric hashes LM) σε μερικές ώρες.

Ο αλγόριθμος δεν υπήρχε σε VHDL , χρειάστηκε να βρούμε τον DES σε Open source κώδικα [***βιβλ. 24***] **και από εκεί να υλοποιήσουμε εμείς τον αλγόριθμο σε γλώσσα για hardware. Η επαλήθευση της ορθής λειτουργίας του αλγόριθμου μας έγινε με κοινές εισόδους τόσο στη δικιά μας σχεδίαση όσο και σε υλοποίηση σε C.**

2.2. MD5 (Message-Digest algorithm 5)

2.2.1. Γενικά στοιχεία

Ο κρυπταλγόριθμος σχεδιάστηκε το 1992 από τον Rivest, [***βιβλ. 20 & 23***] αποτελεί συνέχεια της μονόδρομος hash MD4 λόγω της εύρεσης συγκρούσεων στην MD4 με 2^{20} υπολογισμούς . Προς το παρόν η μόνη αξιόλογη κριτική είναι το μικρό μήκος της σύνοψης που είναι 128 bits, και υπάρχει το ενδεχόμενο επιτυχούς εξαντλητικής αναζήτησης.

Ο MD5 έχει τους εξής στόχους ασφαλείας όπως διατυπώθηκαν από το Rivest

Ασφάλεια. Θα πρέπει να είναι υπολογιστικά αδύνατο να βρεθούν δύο μηνύματα τα οποία να δίνουν το ίδιο αποτέλεσμα σύνοψης.

Άμεση ασφάλεια. Ο αλγόριθμος δε θα βασίζεται σε υποθέσεις, όπως για παράδειγμα στη δυσκολία παραγοντοποίησης ακεραίων.

Ταχύτητα. Ο αλγόριθμος θα είναι βασισμένος σε απλές λογικές πράξεις και ο σχεδιασμός του θα είναι βελτιστοποιημένος για 32-bit αρχιτεκτονικές υπολογιστών.

Απλότητα και κατάληψη μικρού χώρου

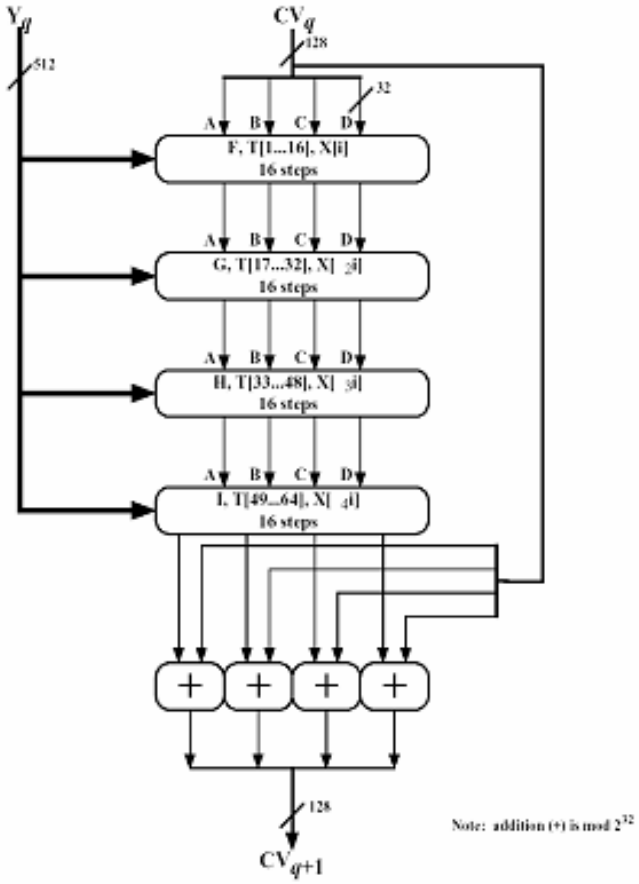
Ο αλγόριθμος θα πρέπει να είναι σχετικά απλός στην περιγραφή του, χωρίς να απαιτεί μεγάλους πίνακες αντικατάστασης τιμών, ή μεγάλα σε μήκος προγράμματα.

Εύνοια αρχιτεκτονικής little-endian. Η αρχιτεκτονική little-endian που είναι βασισμένοι οι επεξεργαστές της Intel x386, αποθηκεύουν το λιγότερα σημαντικό bit σε χαμηλή διεύθυνση μνήμης του byte, σε αντίθεση με την αρχιτεκτονική big-endian που είναι βασισμένοι οι επεξεργαστές Sparc. Έτσι ένας little-endian επεξεργαστής μπορεί να χρησιμοποιεί απ' ευθείας τις αποθηκευμένες δυαδικές λέξεις, ενώ στην περίπτωση του big-endian απαιτείται αντιστροφή. Επειδή γενικά οι big-endian επεξεργαστές είναι γρηγορότεροι στην εκτέλεση πράξεων, θεωρήθηκε ότι η προτίμηση έκφρασης του αλγόριθμου στη μορφή little endian εξισορροπεί τη διαφορά ταχύτητας μεταξύ των δύο οικογενειών επεξεργαστών.

2.2.2. Τεχνικά χαρακτηριστικά του MD5

Ο MD5 δέχεται ως είσοδο ένα μήνυμα αυθαίρετου μήκους και παράγει σύνοψη μήκους 128 bits. Η επεξεργασία γίνεται σε τμήματα των 512 bits, όπου κάθε τμήμα συμμετέχει σε τέσσερις γύρους της συνάρτησης συμπίεσης.

Αρχικά στο μήνυμα προστίθενται bits ώστε το μέγεθος του να είναι ίσο με (448 mod 512). Το προστιθέμενο κομμάτι αποτελείται από μία μονάδα, συνοδευόμενη από τον απαιτούμενο αριθμό μηδενικών (1, 0, 0, ..., 0). Το τελευταίο τμήμα του μηνύματος έχει μήκος 448 bits και τα υπόλοιπα 64 που μένουν για να συμπληρωθεί το μήκος των 512 bits χρησιμοποιείται για να αποθηκευθεί ο αριθμός που εκφράζει το συνολικό μήκος του μηνύματος mod 2^{64}. Τα 512 bits απαιτούν για την αποθήκευση τους 16 δυαδικές λέξεις των 32 bit.



Σχήμα 2.6 MD5 Διεργασία για είσοδο 512 bits

Οι ενδιάμεσες τιμές καθώς και το αποτέλεσμα της σύνοψης αποθηκεύονται σε 4 καταχωρητές A, B, C και D. Κατά την εκκίνηση της διαδικασίας οι καταχωρητές παίρνουν τις ακόλουθες αρχικές τιμές:

A = (67452301)$_{16}$

B = (EFCDAB89)$_{16}$

C = (98BADCFE)$_{16}$

D = (19325476)$_{16}$

Οι καταχωρητές αυτοί ονομάζονται ***αλυσιδωτές μεταβλητές*** (chaining variables). Η συνάρτηση συμπίεσης αποτελείται από τέσσερις γύρους και κάθε γύρος εκτελεί 16 πράξεις. Κάθε πράξη εκτελεί μία μη γραμμική συνάρτηση μεταξύ των τριών από τα A,

B, C και D και προσθέτει το αποτέλεσμα στην τέταρτη μεταβλητή. Τέλος, το αποτέλεσμα αποθηκεύεται σε μια από τις αλυσιδωτές μεταβλητές.

Κάθε γύρος ορίζεται από μια πράξη η οποία εφαρμόζεται 16 φορές. Οι τέσσερις πράξεις είναι:

Γύρος;: a <- b + ((a + $Rj\ (b,\ c,\ d)$ + $M,$+ $T[i])$ <<< $s)$ mod 2^{32} , για 1 ≤j≤ 4,

όπου:

a,b,c,d συνδυασμός των μεταβλητών A, B, C και D. Η σειρά και η αντιστοιχία μεταβάλλεται ανά γύρο και ανά πράξη,

M, η l-στή δυαδική λέξη των 32 bit που αποτελεί το τμήμα των 512 bit

T[i] κατά το *i-στό* βήμα, η τιμή που προκύπτει από το ακέραιο τμήμα της ποσότητας 2^{32} abs(sin(i)), με το *i* σε ακτίνια,

s η ποσότητα κυκλικής ολίσθησης του αποτελέσματος,

Rj( ) η μη γραμμική συνάρτηση του γύρου *j.* Οι τέσσερις μη γραμμικές συναρτήσεις είναι:

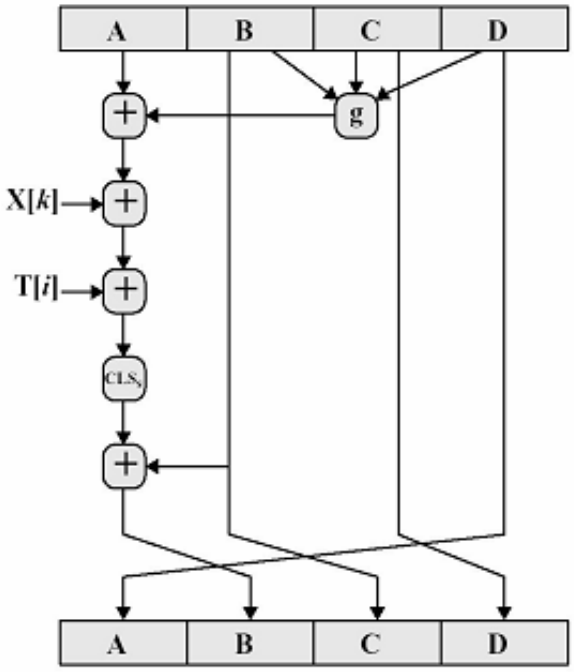
$\mathbf{R_1}$, $(b,c,d) = bc + b'd$

$\boldsymbol{R_2}\ (b,c,d) = bd + c\ d'$

$\boldsymbol{R_3}\ (b,c,d) = b\ xor\ c\ xor\ d$

$\boldsymbol{R_4}\ (b,c,d) = c\ xor\ (b + d')$

Στο τέλος των τεσσάρων γύρων στις τελικές τιμές προστίθενται οι αρχικές τιμές των μεταβλητών A, B, C και D.



Σχήμα 2.7 Συνάρτηση Γύρου MD5

Ο κρυπταλγόριθμος δεν υπήρχε free σε VHDL , οπότε χρειάστηκε να τον μετατρέψουμε από verilog όπου υπήρχε διαθέσιμος σε VHDL ,η επαλήθευση της ορθής λειτουργίας έγινε με test bench τόσο στη δικιά μας υλοποίηση όσο και στην υλοποίηση σε verilog [***βιβλ. 30***]

2.3. Secure Hash Algorithm, SHA-1

Η μονόδρομη συνάρτηση SHA δημοσιεύθηκε το 1993 από το Εθνικό Ινστιτούτο Τυποποίησης και Τεχνολογίας (NIST) και βασίζεται στον MD4, με τη διαφορά ότι η είσοδος δεν μπορεί να είναι μεγαλύτερη των 2^M bits. Επιπλέον, η σύνοψη έχει μέγεθος 160 bits έναντι της σύνοψης της MD4 που έχει μέγεθος 128 bits. [***βιβλ. 22***]

2.3.1. Τεχνικά χαρακτηριστικά του SHA-1

Η αρχική επεξεργασία του μηνύματος είναι ίδια με αυτήν της MD4, δηλαδή στο μήνυμα προστίθεται η ακολουθία (1, 0, 0, ..., 0) έτσι ώστε το συνολικό μέγεθος να είναι ίσο με 448 mod 512. Στη συνέχεια, στα τελευταία 64 bits αποθηκεύεται το μέγεθος του μηνύματος.

Λόγω του μεγαλύτερου μήκους της σύνοψης χρησιμοποιούνται πέντε αλυσιδωτές μεταβλητές, έναντι των τεσσάρων που χρησιμοποιούνται στις MD4 και MD5. Οι μεταβλητές αυτές είναι οι A, B, C, D και E και οι αρχικές τιμές των τεσσάρων πρώτων μεταβλητών είναι όμοιες με αυτές της MD5, ενώ η πέμπτη μεταβλητή έχει την τιμή:

$E = (C3D2E1F0)_{16}$

Παρόμοια με την MD5, το κάθε τμήμα του μηνύματος υποβάλλεται σε τέσσερις γύρους, όπου ο κάθε γύρος αποτελείται από μια μη γραμμική πράξη που εφαρμόζεται 20 φορές. Οι τέσσερις πράξεις της SHA είναι οι ακόλουθες:

$R_1(b,c,d) = bc + bd$ $R_2(b,c,d) = b®c@d$ $R_i(b,c,d) = bc + bd + cd$ $R_4(b,c,d) = b \text{ xor } c \text{ xor } d$

Μια άλλη διαφορά της SHA με την MD5 είναι μια επιπλέον συνάρτηση που επεκτείνει τα 512 bits του τμήματος του μηνύματος σε 2560 bits, ή ισοδύναμα σε 80 δυαδικές λέξεις των 32 bit. Έστω W,, 0 < i < 79 τα τμήματα που προκύπτουν από την επέκταση του τμήματος M,. Η διαδικασία επέκτασης του M, ορίζεται από:

W, = M„ για O < *t*< 15, και

W, = (W,_3 θ $W_{t\_\%}$ θ W,_„ θ W,_16)<< 1, για 16 < t < 79. Τέλος, ο κάθε γύρος είναι της μορφής:

Γύρος;: *(a,b,c,d,e)* «- ((e + *Rj(b,c,d) + S^5(A) + W,+ Kj),a,S^{i0}(b),c,d),* όπου:

η κυκλική ολίσθηση της μεταβλητής *a* κατά *k* bits,

t, ο αύξων αριθμός της πράξης (0 < / < 79),

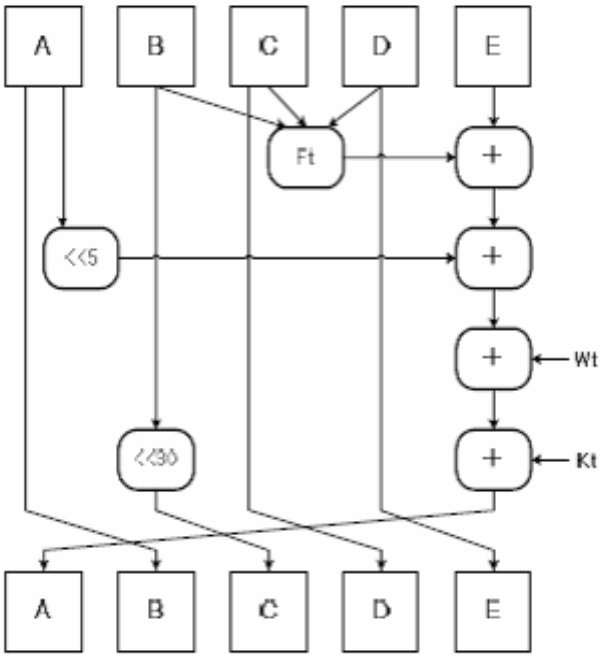
Kj, *μια από τις 4 σταθερές*:

K, = (5A827999),$_6$

K_2 = (6ED9EBA1),$_6$

K_3 = (8F1BBCDC)$_{16}$

K_4 = (CA62ClD6)i$_6$



Σχήμα 2.8 Συνάρτηση Γύρου SHA-1

Και αυτός ο κρυπταλγόριθμος δεν υπήρχε free σε VHDL , οπότε χρειάστηκε να τον μετατρέψουμε από verilog όπου υπήρχε διαθέσιμος σε VHDL ,η επαλήθευση της ορθής λειτουργίας έγινε με test bench τόσο στη δικιά μας υλοποίηση όσο και στην υλοποίηση σε verilog [***βιβλ. 31***]

3. Μοντέλα Επίθεσης

3.1. Επιθέσεις ωμής βίας

Η **brute-force attack** (*επίθεση ωμής βίας*) αναφέρεται στην εξαντλητική δοκιμή πιθανών κλειδιών που παράγουν ένα κρυπτογράφημα, ώστε να αποκαλυφθεί το αρχικό μήνυμα. Τέτοιου είδους επιθέσεις, οι οποίες χρησιμοποιούν όλα τα δυνατά κλειδιά, μπορούν πάντοτε να πραγματοποιηθούν. Συχνά όμως ο επιτιθέμενος ξεκινά την επίθεση χρησιμοποιώντας πιο "πιθανά" κατά την άποψή του κλειδιά, προσπαθώντας με αυτό τον τρόπο να βρει το κλειδί πιο γρήγορα. Πρακτικά, η αναζήτηση σταματάει μόλις βρει το κλειδί, χωρίς να χρειαστεί περαιτέρω ενημέρωση της λίστας κλειδιών. Στην ακαδημαϊκή βιβλιογραφία, η μέθοδος brute-force είναι μέτρο ασφάλειας ενός αλγόριθμου κρυπτογράφησης

Ένας αλγόριθμος κρυπτογράφησης θεωρείται "σπασμένος" αν υπάρχει αλγόριθμος κρυπτανάλυσης ο οποίος μπορεί να βρει το κλειδί με μικρότερη πολυπλοκότητα από τη μέθοδο brute-force, ανεξαρτήτως εάν αυτή η προσπάθεια υπολογισμού είναι εφικτή στην πράξη. Συνήθως, το μήκος των κρυπτογραφικών κλειδιών επιλέγεται με τρόπο τέτοιο, ώστε να απαιτείται υπερβολικά μεγάλος χρόνος υπολογισμών (με βάση τις τρέχουσες υπολογιστικές δυνατότητες) και άρα να μην έχει χρηστική αξία μία τέτοιου είδους επίθεση. [***βιβλ. 32***]

3.2. Επίθεση λεξικών

Μια επίθεση λεξικών (dictionary attack) επιχειρεί να δοκιμάσει "κάθε λέξη στο λεξικό" ως πιθανό κωδικό πρόσβασης για ένα κρυπτογραφημένο μήνυμα.

Μια επίθεση λεξικών είναι γενικά αποδοτικότερη από μια επίθεση ωμής βίας επειδή οι χρήστες επιλέγουν χαρακτηριστικά τους φτωχούς κωδικούς πρόσβασης. Οι επιθέσεις λεξικών είναι γενικά μακριά λιγότερο επιτυχείς ενάντια στα συστήματα που χρησιμοποιούν τα pass-phrases αντί των κωδικών πρόσβασης.

Υπάρχουν δύο μέθοδοι την επιτυχία μιας επίθεσης λεξικών. Η πρώτη μέθοδος την επιτυχία μιας επίθεσης λεξικών είναι να χρησιμοποιηθεί ένα μεγαλύτερο λεξικό, ή περισσότερα λεξικά. Τα τεχνικά λεξικά και τα λεξικά ξένης γλώσσας θα αυξήσουν τη γενική πιθανότητα το σωστό κωδικό πρόσβασης.

Η δεύτερη μέθοδος την επιτυχία μιας επίθεσης λεξικών είναι να εκτελεσθεί ο χειρισμός σειράς στο λεξικό. Παραδείγματος χάριν, το λεξικό μπορεί να έχει τη λέξη "κωδικός πρόσβασης" «password». Οι κοινές τεχνικές χειρισμού σειράς θα δοκιμάσουν τη λέξη προς τα πίσω (drowssap), με τις κοινές αντικαταστάσεις αριθμός-επιστολών (p4ssw0rd), ή με τη διαφορετική κεφαλαιοποίηση (κωδικός πρόσβασης).

Φυσικά, τα πολύ μικρά λεξικά μπορούν να οδηγήσουν στη γρηγορότερη επιτυχία, εάν ένας ή περισσότεροι από τους στόχους κρυπτογραφούνται με έναν πολύ αδύνατο κωδικό πρόσβασης. Ένας κατάλογος επίλεκτων υποψηφίων των ονομάτων ανθρώπων μπορεί να παραγάγει τα καταπληκτικά αποτελέσματα. Ένα λεξικό των πιθανών κωδικών πρόσβασης είναι ακριβέστερα γνωστό ως κατάλογος λέξεων. Εάν μια εκτενής επίθεση λεξικών αποτυγχάνει, μπορεί να είναι σημαντικό να προσφύγει σε μια επίθεση ωμής βίας. Μια επίθεση ωμής βίας θα επιτύχει τα αποτελέσματα τελικά από μια επίθεση λεξικών. [***βιβλ. 32***]

3.3. Πίνακας ουράνιων τόξων - Rainbow tables

Ένας **πίνακας ουράνιων τόξων** είναι ένας look up table(πίνακας αναζήτησης) που προσφέρει time – memory trade off (ανταλλαγή χρόνου/μνήμης) και χρησιμοποιείται στην ανάκτηση ενός κωδικού πρόσβασης από ένα hash password (Κατακερματισμένο κωδικό) που έχει παραχθεί από μια hash function(συνάρτηση κατακερματισμού) συνήθως κρυπτογραφική [***βιβλ. 8***]

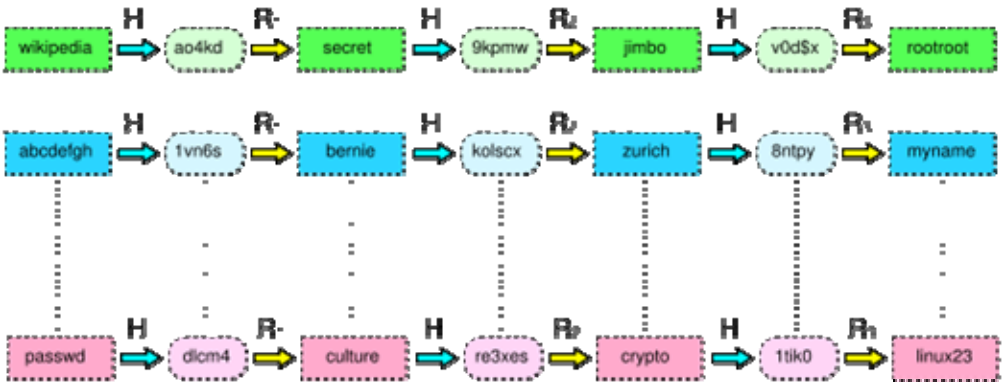
3.3.1 Ανταλλαγή χρόνου/μνήμης

Στην πληροφορική όταν αναφερόμαστε σε **space-time** ή **time–memory trade off** (ανταλλαγή χρόνου/μνήμης) είναι μια κατάσταση όπου η χρήση μνήμης μπορεί να μειωθεί με κόστος την πιο αργή εκτέλεση προγράμματος, ή αντίστροφα, ο χρόνος υπολογισμού μπορεί να μειωθεί με κόστος την αυξανόμενη χρήση μνήμης.

Δεδομένου ότι οι σχετικές δαπάνες των κύκλων της CPU, ο χώρος της RAM, και ο χώρος του σκληρού δίσκου αλλάζουν , με κατάλληλες επιλογές για space-time μπορεί να αλλάξει και η πολυπλοκότητα του προβλήματος προς όφελος. Έτσι π.χ ένας αλγόριθμος που περιλαμβάνει έναν look-up table ολόκληρο, μπορεί να μειώσει το χρόνο υπολογισμού, αλλά να αυξήσει το ποσό μνήμης που απαιτείται, ή να αυξήσει το χρόνο υπολογισμού, αλλά με μείωση των απαιτήσεων μνήμης.

Μια κοινή εφαρμογή των rainbow tables είναι να καταστούν εφικτές οι επιθέσεις ενάντια στους κωδικούς πρόσβασης. Για να καταστήσει αυτήν την επίθεση δυσκολότερη, ως και απραγματοποίητη οι κωδικοί πρόσβασης χρησιμοποιούν salt.

To salt περιλαμβάνει τα τυχαία bits που χρησιμοποιούνται ως μια από τις εισόδους σε μια κρυπτογραφική συνάρτηση ενώ η άλλη είσοδος είναι συνήθως ένας κωδικός πρόσβασης. Το salt μπορεί επίσης να χρησιμοποιηθεί ως κλειδί σε ένα κρυπτογραφικό αλγόριθμο. Το salt κρατιέται μερικές φορές μυστικό , αυτό παρέχει ένα πλεονέκτημα όταν κλέβεται μια βάση δεδομένων κωδικού πρόσβασης, στην οποία το salt δεν φαίνεται.



3.1. Ένας πίνακας ουράνιων τόξων

3.3.2 Ανάλυση Πινάκων Ουρανίου Τόξου

Ένας πίνακας ουράνιων τόξων είναι μια αναπαράσταση ακολουθιών(αλυσίδων) κωδικών πρόσβασης. Κάθε αλυσίδα αρχίζει με έναν *αρχικό κωδικό πρόσβασης*, ο οποίος περνά μέσω μιας hash function κρυπτογραφίας ***H***. Το προκύπτον hash password τροφοδοτείται έπειτα σε μια reduction function ***R*** **–** συνάρτηση ανακατασκευής , η οποία παράγει έναν διαφορετικό κωδικό πρόσβασης. Η διαδικασία επαναλαμβάνεται έπειτα για έναν σταθερό αριθμό επαναλήψεων. Ο αρχικός κωδικός πρόσβασης και ο τελευταίος κωδικός της αλυσίδας που παράγεται από την R αποτελούν μια είσοδο σε ένα πίνακα ουράνιων τόξων.

Η ανάκτηση ενός κωδικού πρόσβασης που χρησιμοποιεί έναν πίνακα ουράνιων τόξων είναι μια διαδικασία δύο βημάτων.

1ο βήμα: Ο hash password χρησιμοποιείται για την δημιουργία αλυσίδας μέσω της reduce-hash ακολουθίας. Η δομή του πίνακα και η λειτουργία της reduction function εγγυώνται ότι το τρέχων hash θα ταιριάζει με τελικό hash με μια από τις αλυσίδες που περιέχονται ήδη στα rainbow tables .

2ο βήμα: Η ακολουθία δημιουργίας αλυσίδας επαναλαμβάνεται αρχίζοντας από αυτόν τον αρχικό κωδικό πρόσβασης έως ότου βρεθεί το αρχικό hash. Ο κωδικός πρόσβασης που χρησιμοποιείται στην τελευταία επανάληψη *είναι* ο κωδικός πρόσβασης που ανακτάται.

Το περιεχόμενο των rainbow tables δεν εξαρτάται από την είσοδο του αλγορίθμου. Δημιουργείται μια φορά και έπειτα χρησιμοποιείται επανειλημμένα για τον έλεγχο.

Η αύξηση του μήκους της αλυσίδας μειώνει το μέγεθος του πίνακα. Αυξάνει το χρόνο που απαιτείται για να κατασκευασθεί μια αλυσίδα, και αυτό είναι **η ανταλλαγή χρόνου/μνήμης του πίνακα ουράνιων τόξων**. Σε μια απλή περίπτωση αλυσίδας ενός μόνο στοιχείου , η αναζήτηση είναι πολύ γρήγορη, αλλά ο πίνακας είναι πολύ μεγάλος. Όσο οι αλυσίδες μεγαλώνουν , η αναζήτηση επιβραδύνεται , αλλά το μέγεθος του πίνακα μικραίνει.

Το τελικό αποτέλεσμα είναι ένας πίνακας που περιέχει τη στατιστικά υψηλότερη πιθανότητα για ανεύρεση ενός κωδικού πρόσβασης εντός ενός μικρού χρονικού διαστήματος. Η πιθανότητα επιτυχίας του πίνακα εξαρτάται από τις παραμέτρους που χρησιμοποιούνται για να τον παραγάγουν. Αυτοί περιλαμβάνουν το σύνολο των χαρακτήρων που μπορεί να υπάρχουν σε ένα κωδικό πρόσβασης , το μήκος του κωδικού πρόσβασης, το μήκος των αλυσίδων, καθώς και ο αριθμός των πινάκων. [***βιβλ. 13***]

4. Rainbow tables για τους Κρυπταλγόριθμους LMhash-MD5-SHA1

4.1. Εισαγωγικά

Αν $E : \{0, 1\}^n \times \{0, 1\}^k \rightarrow \{0, 1\}^n$ είναι ένα κρυπτογραφημένο τμήμα με μήκος εισόδου n και κλειδιού μήκους k. Θα εξετάσουμε τον LM hash με 56 bits είσοδο . Η κρυπτογράφηση δείχνεται ως C = EK(P) όπου C, K και P το κρυπτοκείμενο , το κλειδί και η είσοδος το απλό κείμενο . Για ένα σταθερή και γνωστή είσοδο P , η χαρτογράφηση EK(P) είναι μια μονόδρομη συνάρτηση από το κλειδί στο κρυπτοκείμενο . Για την ανταλλαγή χρόνου-μνήμης δυο συναρτήσεις συνήθως ορίζονται. Η πρώτη είναι η g :

$\{0, 1\}^{64} \rightarrow \{0, 1\}^{56}$ η οποία αντιστοιχεί ένα κρυπτοκείμενο σε μια K bits σειρά οπότε μπορούμε να γράψουμε :

$$g(C) = g(C1, C2, \ldots, C64) = (X1, X2, \ldots, X56)$$

Αυτή η πράξη συνήθως καλείται ως mask function ή reduction function. Υπάρχουν πολλές πιθανότητες να οριστεί αυτή η πράξη : κάποιοι προτείνουν να αφήσουμε 8 bit και να μεταλλάξουμε τα άλλα 56, οπότε και έχουμε σαν αποτέλεσμα άλλες 2^{280} περιπτώσεις. Άλλες επιλογές που είναι ποιο κατάλληλες για hardware υλοποιήσεις περιλαμβάνουν ανταλλαγές bit , συναρτήσεις Xor και τα λοιπά . Το περιγράφουμε αναλυτικά στο κεφάλαιο 5.1

Δεύτερο ορίζουμε μια πράξη $f : \{0, 1\}^{56} \rightarrow \{0, 1\}^{56}$ που αντιστοιχεί το χώρο του κλειδιού στον εαυτό του:

f(K) = g(EK(P)) = g(C1, C2, . . . , C64) = g(C), για κάθε K ανήκει $\{0,1\}^{56}$.

Αυτή η κατασκευή έχει τις ρίζες της στο Hellman και γενικεύτηκε από τους Kusuda και Matsamoto . Με την αντικατάσταση των κρυπτοκειμένων με κλειδιά, μια αλυσίδα μπορεί να κατασκευαστεί

$$K_i \xrightarrow{E_{K_i}(P)} C_i \xrightarrow{g(C_i)} K_{i+1}\,,$$

η οποία μπορεί να γραφτεί ως αλυσίδα κλειδιών

$$K_i \xrightarrow{f} K_{i+1} \xrightarrow{f} K_{i+2}\,.$$

Στον αρχικό αλγόριθμο του Hellman κατασκευάζονται m αλυσίδες μήκους t . Μια αλυσίδα αποθηκεύει μόνο το πρώτο και το τελευταίο στοιχείο από κάθε αλυσίδα σε ένα πίνακα. Δεδομένου ενός κρυπτοκειμένου C με ένα γνωστό αρχικό κείμενο κάποιος μπορεί να προσπαθήσει να βρει ένα κλειδί που χρησιμοποιήθηκε για να παραχθεί το C κατά τον ακόλουθο τρόπο. Οι αλυσίδες (μέχρι ενός σταθερού μήκους t) ψάχνονται μέχρι ένα κλειδί που να είναι ταυτόσημο με το τελευταίο κλειδί κάποιας αλυσίδας βρεθεί. Χρησιμοποιώντας το πρώτο κλειδί η αλυσίδα μπορεί να ανακατασκευασθεί και το σωστό κλειδί είναι αυτό ακριβώς πριν από το C. Το τυπικό μέγεθος των παραμέτρων για ένα K-bit κλειδί είναι t = m = $2^{k/3}$. Αν κάποιος χρησιμοποιήσει r=$2^{k/3}$ πίνακες ο συνολικός χρόνος προϋπολογισμού είναι 2^k εκτιμήσεις από f και πρέπει να αποθηκευτούν $2^{k/3}$ τιμές των 2k bits . Η ανάνηψη ενός κλειδιού απαιτεί $2^{k/3}$ εκτιμήσεις του f. Η πιθανότητα επιτυχίας αυτής της μεθόδου εξαρτάται από τον αριθμό των επαναλαμβανόμενων στοιχείων στις αλυσίδες . Επαναλήψεις συμβαίνουν λόγο συγχωνεύσεων κάποιων αλυσίδων και λόγο ότι κάποιες αλυσίδες μπαίνουν σε Loop. Για το τυπικό μέγεθος παραμέτρου t= m = r = $2^{k/3}$, η πιθανότητα επιτυχίας είναι περίπου 0,55

Η προσέγγιση με διακριτά σημεία αποφεύγει μια αναζήτηση σε πίνακα μετά από κάθε υπολογισμό της συνάρτησης , καθώς ένας αποδοτικός τρόπος εφαρμογής της αναζήτησης σε ένα μεγάλο πίνακα θα ήταν πολύ ακριβός. Ένα διακριτό σημείο είναι ένα κλειδί που έχει μια ιδιότητα που είναι εύκολο να αναγνωριστεί (πχ τα 20 πιο σημαντικά bit είναι μηδέν) αυτό σημαίνει ότι κάποιος χρειάζεται να ελέγξει μετά από κάθε επανάληψη εάν μια τιμή είναι ένα διακριτό σημείο ή όχι. Οι αλυσίδες κατασκευάζονται αρχίζοντας και τελειώνοντας με ένα διακριτό σημείο: επίσης αυτό

επιτρέπει τη μείωση του αποθηκευτικού χώρου ανα αλυσίδα και τον έλεγχο για συγχωνευμένη αλυσίδα (αλλά η ρίψη τέτοιας αλυσίδας υπονοεί πως κάποιος χρειάζεται να αυξήσει το χρόνο προϋπολογισμού. Όμως στη μεταβλητή του διακριτού σημείου οι αλυσίδες είναι διαφορετικές σε μήκος και θα έχουν μεγαλύτερη πιθανότητα να συγχωνευθούν (μειώνοντας την πιθανότητα επιτυχίας της επίθεσης.)

Οι πίνακες που προτάθηκαν από τον Oechslin χρησιμοποιούν διαφορετική συνάρτηση g για κάθε επανάληψη. Πιο αναλυτικά οι αλυσίδες έχουν ένα σταθερό μήκος t και χρησιμοποιούν διαφορετικές mask function ενός μιας αλυσίδας : $g_1, \ldots, g_t$

Ξεκινάει από την προτελευταία στήλη και εφαρμόζει g_t μετά από την προ-προ-τελευταία και εφαρμόζει το g_{t-1} και g_t μέχρι που φτάνει τη πρώτη και εφαρμόζει από g_1 έως g_t

Ο συνολικός αριθμός των επαναλήψεων είναι t(t-1)/2 . Αυτή η διαδικασία μας επιτρέπει να μειώσουμε τις προσβάσεις μνήμης άλλα ταυτόχρονα μειώνει την πιθανότητα της συγχώνευσης αλυσίδων. 2 αλυσίδες θα συγχωνευτούν αν τα 2 σημεία συγχώνευσης είναι στην ίδια θέση σε μια αλυσίδα . Λόγω της μειωμένης πιθανότητας να γίνει αυτό οι πίνακες του Oechslin μπορεί να είναι πολύ μεγαλύτεροι , τυπικά μόνο μερικοί πίνακες είναι αναγκαίοι . Η μέθοδος έχει εφαρμοστεί σε λογισμό (για τους κωδικούς των Windows) ενώ μόνο μια εφαρμογή έχει δημοσιευθεί για υλοποίηση σε hardware και αφορά τον DES [***βιβλ. 9***]

4.2. Θεωρητικοί παράμετροι για τους κρυπταλγόριθμους

Εδώ θα λάβουμε υπόψιν την εφαρμογή της ανταλλαγής χρόνου-μνήμης για τους κρυπταλγόριθμους MD5, SHA1, LMhash.

Και για τον LMhash επιλέγουμε είσοδο 56 bits μιας και το βασικό χαρακτηριστικό του είναι ότι τα 14 bytes χωρίζονται σε δύο μισά συνεπώς αρκεί να δημιουργήσουμε

ένα table για τα 7bytes και θα χρησιμοποιήσουμε το ίδιο για το άλλο μισό του κωδικού

Για τον MD5 και στον SHA1 μιας και είναι ασύμμετροι κρυπταλγοριθμοι μεταβλητής εισόδου επιλέγουμε εμείς ως σύμβαση να δεχτούμε ότι η είσοδος θα είναι και εδώ 56 bits μιας και όπως θα εξηγήσουμε παρακάτω είναι αδύνατο σχεδόν να δημιουργήσουμε rainbow tables με αυξημένο ρυθμός επιτυχίας. Επειδή όμως η επεξεργασία του MD5 και του SHA1 γίνεται ανα 512 bits θα ακολουθήσουν 392 bits λόγω (392+56) 448 mod 512 =448 bits . Συνεπώς θα προσθέσουμε ένα '1' και 391 bits ' 0'. Τέλος θα προσθέσουμε 64 bits το οποίο αναπαριστά το αρχικό μήκος του εισαγόμενου κειμένου.

Ο Lmhash όπως αναφέραμε και πιο πάνω κατά την περιγραφή του μπορεί να αποτελείται, από κεφαλαία γράμματα και αριθμούς , δηλαδή 26 +10 =36 διαφορετικοί χαρακτήρες για κάθε ψηφίο του. Τα 36 αναπαρίστανται με 6 bits πληροφορίας το οποίο έχει ως αποτέλεσμα για το εύρος του κλειδιού 42 bit , συνεπώς μπορεί να ανακτηθεί $2^{48*2/3}$ αξιολογήσεις συναρτήσεων

Ο MD5 και ο SHA1 , όπως αναφέραμε πιο πάνω μπορούν να πάρουν οποιαδήποτε είσοδο όμως για λόγους ευκολίας δικούς μας θεωρήσαμε ως πιθανές εισόδους τα μικρά γράμματα – κεφαλαία – αριθμούς καθώς και 2 χαρακτήρες ¨ \ ¨ και ¨ . ¨ Συνεπώς 26+26+10+2=64 τα οποία αναπαρίστανται ομοίως με 6 bit πληροφορίας και συνεπώς ισχύουν τα ίδια με την παραπάνω περίπτωση.

4.3. Όρια και παράμετροι

Θα παρουσιάσουμε μερικούς συμβολισμούς. Ας υποθέσουμε ότι t είναι το μήκος των αλυσίδων και m να συμβολίσουμε τον αριθμό των αλυσίδων σε κάθε πίνακα και r τον αριθμό των πινάκων . Αυτές οι παράμετροι μπορούν να ποικίλουν με σκοπό να

συντονίσουν το ρυθμό επιτυχίας καθώς η ανταλλαγή χρόνου-μνήμης είναι μέθοδος που βασίζεται στη πιθανότητα.

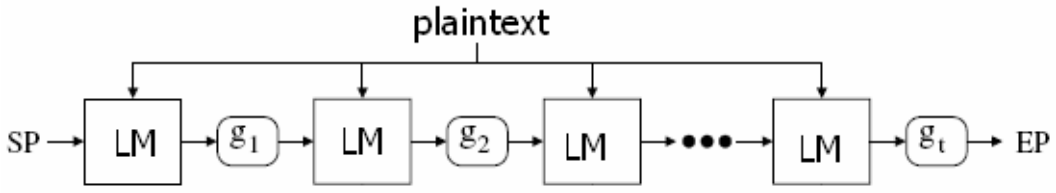
Το σχηματικό διάγραμμα μιας αλυσίδας και η ολική δομή δείχνετε στα επόμενα σχήματα

Τα όρια της μνήμης M (χρησιμοποιείται για την αποθήκευση των πινάκων) και το χρόνο T (απαιτούμενος χρόνος για εύρεση του κωδικού αρχίζοντας από την hash) είναι τα ακόλουθα

$M= m * r * m_0$

$T= t * r * t_0$

Εδώ , m_0 είναι το μέγεθος της μνήμης που απαιτείται για να αποθηκευτεί κάθε αλυσίδα δηλαδή το SP και το EP.Στην περίπτωση μας είναι 14 byte (112 bits) . Ομοίως το t_0 είναι ο χρόνος έχουμε τη παραγωγή ενός κρυπτοκειμένου από ένα αρχικό κείμενο .



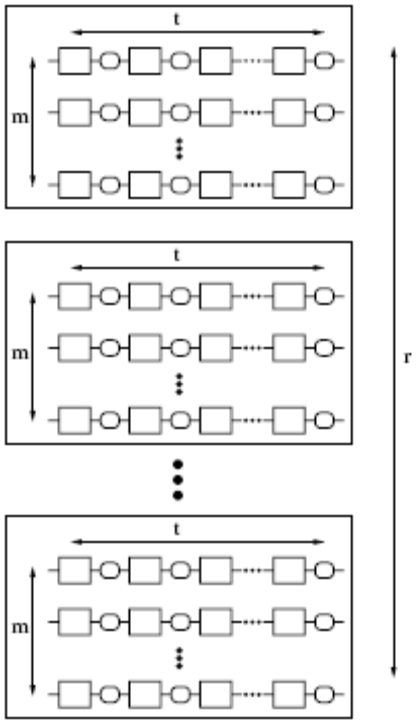
Σχήμα 4.1. Σχηματική αναπαράσταση της αλυσίδας Ουρανίου τόξου .

Ο ρυθμός επιτυχίας ενός μονού πίνακα σύμφωνα με τον Oechslin μπορεί να υπολογιστεί όπως φαίνεται παρακάτω :

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

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

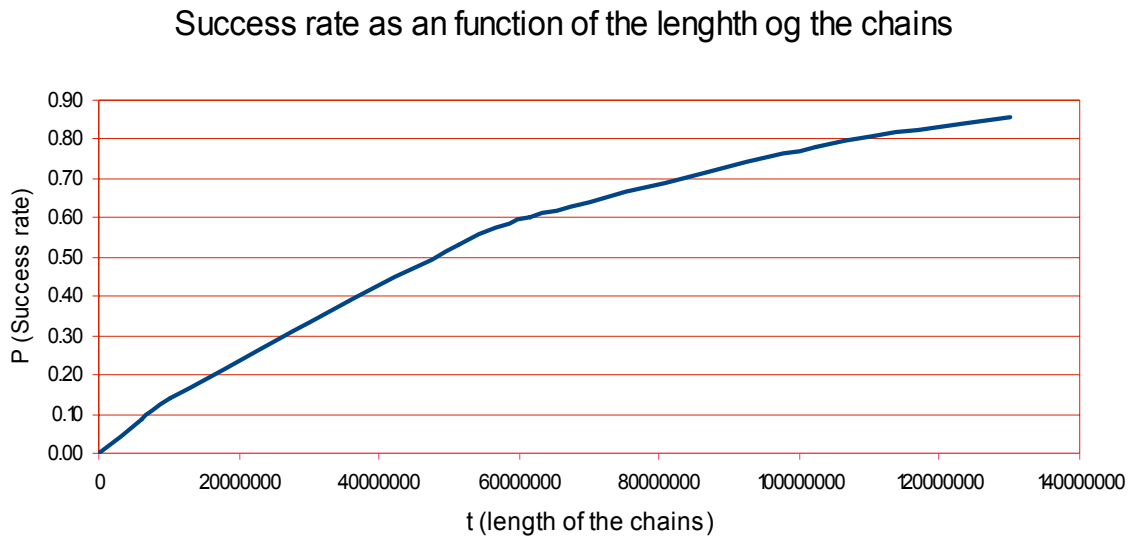
Στο σχήμα 4.3 ο ρυθμός επιτυχίας φαίνεται συνάρτηση της αλυσίδας t . Είναι προφανές ότι η πιθανότητα αυξάνεται γρηγορότερα στην αρχή με το όσο αυξάνεται το μήκος των αλυσίδων.



Σχήμα 4.2. Σχηματική αναπαράσταση της δομής των πινάκων ουρανίου τόξου

Ο ρυθμός επιτυχίας για πολλαπλούς πίνακες μπορεί να υπολογιστεί από:

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



Σχήμα 4.3. Ο ρυθμός επιτυχίας συνάρτηση της αλυσίδας t

m	N	t	P rainbow = 1-(1-m/N)^t
65536	2^42	5	0.00
65536	2^42	1,000	0.09
65536	2^42	10,000	0.14
65536	2^42	51,200,000	0.53
65536	2^42	61,440,000	0.60
65536	2^42	102,400,000	0.78
65536	2^42	130,000,000	0.86

Πίνακας 4.1 Υπολογισμός ρυθμού επιτυχίας

Εδώ εξετάζουμε μόνο ένα πίνακα, στη εργασία του Oechslin τα καλύτερα αποτελέσματα κρυπτογραφημάτων επιτεύχθηκαν χρησιμοποιώντας μόνο 5 πίνακες.

5 . ΥΛΟΠΟΙΗΣΗ ΣΕ HARDWARE

Εδώ θα εξετάσουμε με λεπτομέρεια στην εφαρμογή των προϋπολογισμών των πινάκων στο hardware , περιγράφουμε το σχεδιασμό και δίνουμε μια εκτίμηση της απόδοσης συγκρίνοντας με software.

5.1 Η δική μας λύση σε FPGA

Η πιο κρίσιμη επιλογή σχετίζεται με τις mask function , είναι μια πράξη μείωσης των bits που αντιστοιχεί ένα κρυπτοκείμενο σε ένα κλειδί . Υπάρχουν διάφορες επιλογές , αναφέρουμε κάποιες από αυτές :

- μεταλλάξεις π.χ. S-boxes
- πράξεις Xor
- ανταλλαγές bit

Καθώς ενδιαφερόμαστε σε εφαρμογές hardware , είναι σημαντικό να επιλεχθούν κατάλληλες mask functions με χαμηλή πολυπλοκότητα για το hardware. Από αυτή την οπτική , και οι τρεις προτεινόμενες επιλογές είναι κατάλληλες . Όμως για πίνακες ουρανίου τόξου μια αλυσίδα περιέχει διαφορετικές μάσκες συνεπώς η λογική ελέγχου πρέπει να ελαττωθεί . Με βάση αυτό xor- ing είναι η πλέον κατάλληλη λύση.

Επιπλέον στη δίκη μας περίπτωση οι μεταλλάξεις μπορεί να μη προσφέρουν αρκετές λύσεις για διαφορετικές μάσκες. Εφόσον η πολυπλοκότητα είναι 2^{42} επιλέγουμε να απαλλαγούμε από τα τελευταία 14 bits και να κάνουμε xor με 42 bit counter.

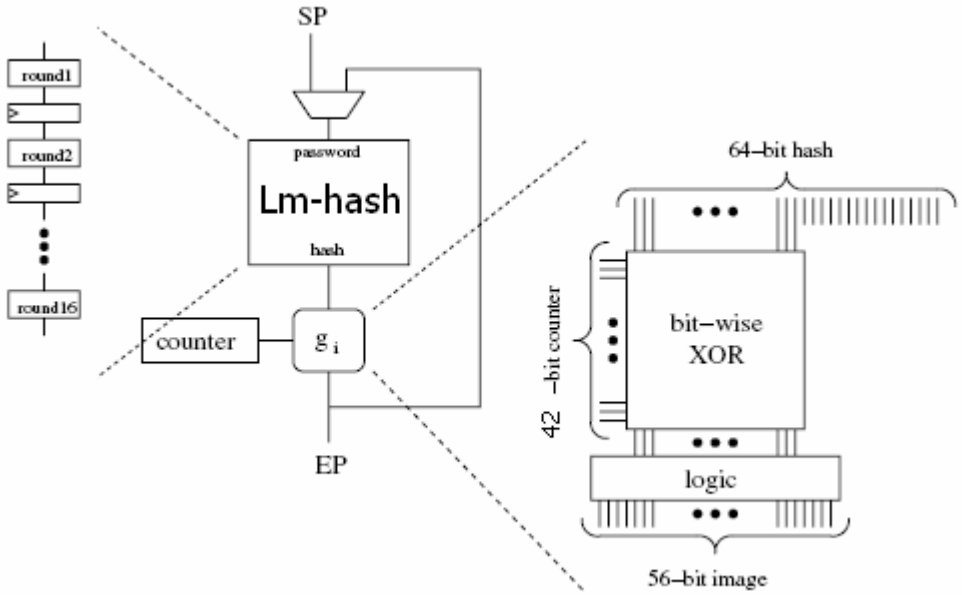
Με αυτόν τον απλό τρόπο, μπορούμε να χρησιμοποιήσουμε μόνο μια generic mask function η οποία διαφέρει με τις διαφορετικές καταστάσεις του μετρητή έχοντας σαν αποτέλεσμα 2^{42} διαφορετικές mask function.

Τέλος το 42bit output της πράξης xor περνά μέσα από μερικές λογικές πύλες για να αποκτήσει ένα 56 bit αποτέλεσμα το οποίο είναι ένα έγκυρο 56bit κλειδί (περιέχει

μόνο κεφαλαία και αριθμούς για τον Lmhash) (κεφαλαία , μικρά , αριθμούς , ¨\¨ , ¨.¨ για τους SHA1 και MD5)

Το παρακάτω σχέδιο απεικονίζει την αρχιτεκτονική μας για τη δημιουργία αλυσίδων πινάκων ουρανίου τόξου. Για να κατασκευασθεί μια αλυσίδα μια εναλλακτική ακολουθία από μπλοκ (αρχικού κειμένου - κρυπτοκειμένου) και mask function εφαρμόζεται . Αυτό γίνεται χρησιμοποιώντας ένα βρόγχο ανατροφοδότησης.

Η δημιουργία SP(Start Point) δηλαδή αρχικών κλειδιών υλοποιείται και αυτή σε Hardware προκειμένου να συνεισφέρουν σε ποιο αποδοτικό προϋπολογισμό. Ποιο συγκεκριμένα , φορτώνοντας SP έξω από την FPGA θα δημιουργήσει ένα πλεόνασμα χρόνου, που είναι ο χρόνος επικοινωνίας FPGA – PC. Δημιουργούμε δλδ ένα counter για να παράγουμε Start points.



Σχήμα 5.1. Αρχιτεκτονική pipeline για την δημιουργία αλυσίδων ουρανίου τόξου

Τα ζευγάρια αρχικού-τελικού κειμένου αποθηκεύονται σε ένα πίνακα κατακερματισμού όπου τα EP λειτουργούν σα δείκτες, ο λόγος είναι τα αποτελέσματα να είναι ταξινομημένα βάση του EP

Λόγω ότι τα EP είναι 56 bits και η μνήμη μας έχει βάθος έως 2^{16}(το μέγιστο που μπορούμε να παράγουμε σε VIRTEX5 για πλάτος 112 bits) πρέπει με κάποιο τρόπο τα 56 bits να αντιστοιχούνται σε 16 η και λιγότερα (στη παραλληλία που θα αναπτύξουμε παρακάτω πολλαπλασιάζουμε τις μνήμες μικραίνοντας το βάθος άρα η αντιστοίχηση δε θα ναι 56->16 αλλά σε λιγότερα) Για το λόγο αυτό χρησιμοποιήσαμε μια hash function ώστε να υπάρχει μια αντιστοίχηση με καλή διασπορά αποτελεσμάτων .

Αυτό όμως έχει να κίνδυνο να υπάρχουν θέσεις στη μνήμη κενές λόγω ότι στη αντιστοίχηση δεν θα πιαστούν όλες οι τιμές σε αντίθεση με την αρχική μας σχεδίαση χρησιμοποιήσαμε 2 μνήμες και την εγγραφή στη μνήμη γινόταν σειριακά , χωρίς τα αποτελέσματα να είναι sorted. Λογικά ο χρόνος να ταξινομηθούν σε software ήδη ταξινομημένοι πίνακες (μιας και οι πίνακες που παράγουμε είναι μικροί σε βάθος , άρα ο στόχος μας είναι να παράγουμε πολλούς και να τους στέλνουμε στο pc στο δίσκο όπου και θα ταξινομούνται) θα ναι πολύ λιγότερος) συνεπώς η λογική της ταξινόμησης βάσει EP σε hardware κερδίζει και άλλο χρόνο από το να κάναμε την ταξινόμηση στο PC

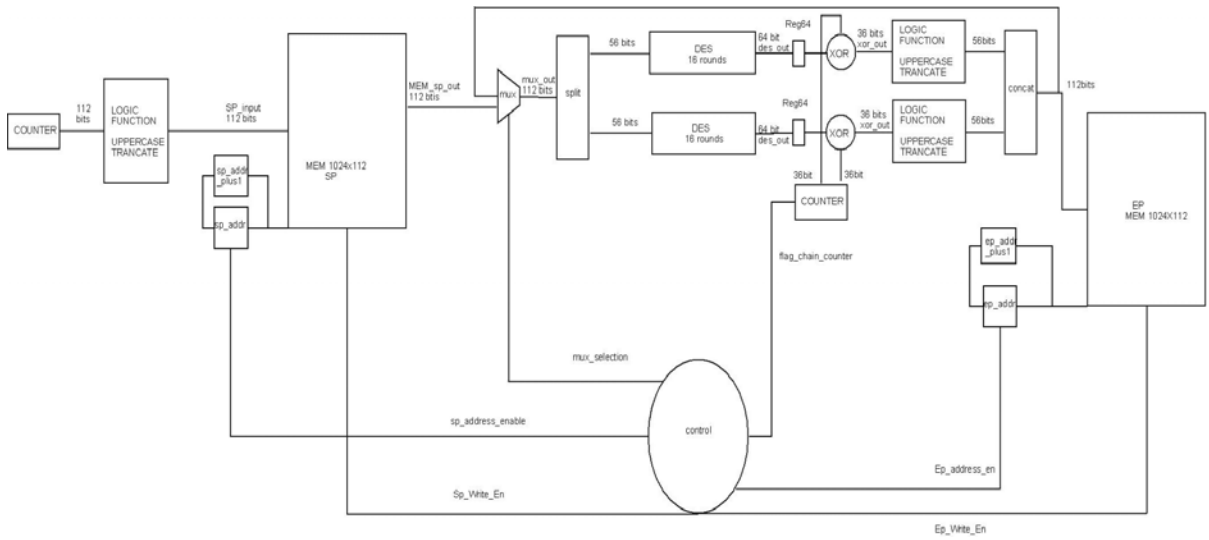
Αφού ταξινομηθούν οι εισαγωγές στους πίνακες το On-line κομμάτι πρέπει να πραγματοποιηθεί . Ανατρέχει στις τιμές της εξόδου της FPGA μέχρι το κλειδί να βρεθεί. Ανακτώντας ένα μεμονωμένο κλειδί σε μια αλυσίδα ουρανίου τόξου παίρνει περίπου t(t-1)/2 εκτιμήσεις πράξεων . Αυτό το κομμάτι μπορεί να γίνει στο software η για να γίνει πιο γρήγορα σε μια FPGA , μια Pipeline αναζήτηση μπορεί να γίνει σε λίγα λεπτά.

Υπάρχουν 2 περιπτώσεις όταν βρίσκεται ότι ένα SP δεν οδηγεί στο σωστό κλειδί , αυτές συνήθως αναφέρονται ως ψευδείς συναγερμοί. Πρώτα μπορεί, το κλειδί να είναι μέρος μιας αλυσίδας με το ίδιο endpoint αλλά δεν είναι στο πίνακα. Η δεύτερη περίπτωση ψευδούς συναγερμού συμβαίνει όταν ένα κλειδί είναι σε μια αλυσίδα η οποία συγχωνεύεται με μια άλλη αλυσίδα στο πίνακα. Για τις αλυσίδα ουρανίου τόξου η συγχώνευση θα συμβεί όταν και μόνο αν η σύγκρουση συμβεί στην ίδια θέση σε 2 αλυσίδες. Τις αλυσίδες μήκους t η πιθανότητα μια σύγκρουση να είναι συγχώνευση είναι μόνο 1/t . Όπως σημειώνεται από το Oechslin είναι πιθανό να δημιουργηθούν

πίνακες χωρίς καθόλου συγχώνευση. Όμως αυτό επιλύει μόνο τη περίπτωση ψευδούς συναγερμού και παραμένει το πρόβλημα να δημιουργηθούν πίνακες οι οποίοι καλύπτουν το χώρο του κλειδιού το περισσότερο δυνατόν.

5.2 Βελτιστοποιήσεις

Στην αρχική μας υλοποίηση για τον Lm hash τα EP και SP αποθηκεύονταν σε 2 διαφορετικές μνήμες, έτσι οι πίνακες γεμίζαν σειριακά , χωρίς να είναι ταξινομημένοι. Παράλληλα το κλειδιά που δημιουργούσαμε σε μια rainbow αλυσίδα ήταν 112bits , λόγω ότι ο Lmhash είναι 14 bytes.
Η αρχική μας σχεδίαση φαίνεται στο παρακάτω σχήμα.



Σχήμα 5.2 Υλοποίηση Πίνακα Ουρανίου Τόξου για τον Lmhash

Παρατηρούμε στο σχήμα ότι η σχεδίαση μας αποτελείται από 4 μέρη :
1)Το Sp_init το οποίο περιλαμβάνει το counter που παράγει τα αρχικά SP , κάποιες λογικές συναρτήσεις ελέγχου ώστε αυτό που παράχθηκε από τον counter είναι

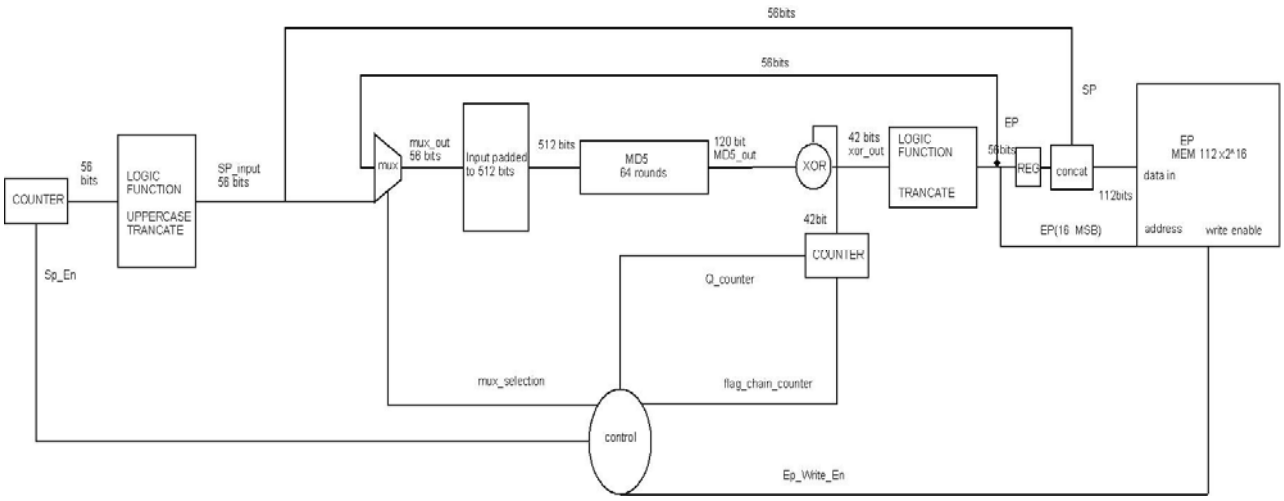
συμβατό με το σύνολο των δυνατών χαρακτήρων που μπορεί να έχει ως είσοδο ο αλγόριθμος , την μνήμη , και ένα δείκτη διευθυνσιοδότησης της μνήμης

2) Το Rainbow_chain που αποτελείται από τον αλγόριθμο που παράγει το hash κρυπτοκείμενο , ένα counter που παράγει 42 bit ποσότητες για να γίνουν xor με το αποτέλεσμα του αλγορίθμου, μια σειρά λογικών πράξεων , έλεγχου ώστε το αποτέλεσμα να είναι συμβατό με το σύνολο των δυνατών χαρακτήρων που μπορεί να έχει ως είσοδο ο αλγόριθμος , και ένα mux που επιλέγει την είσοδο στον αλγόριθμο αν είναι το SP η το αποτέλεσμα από τον αλγόριθμο .

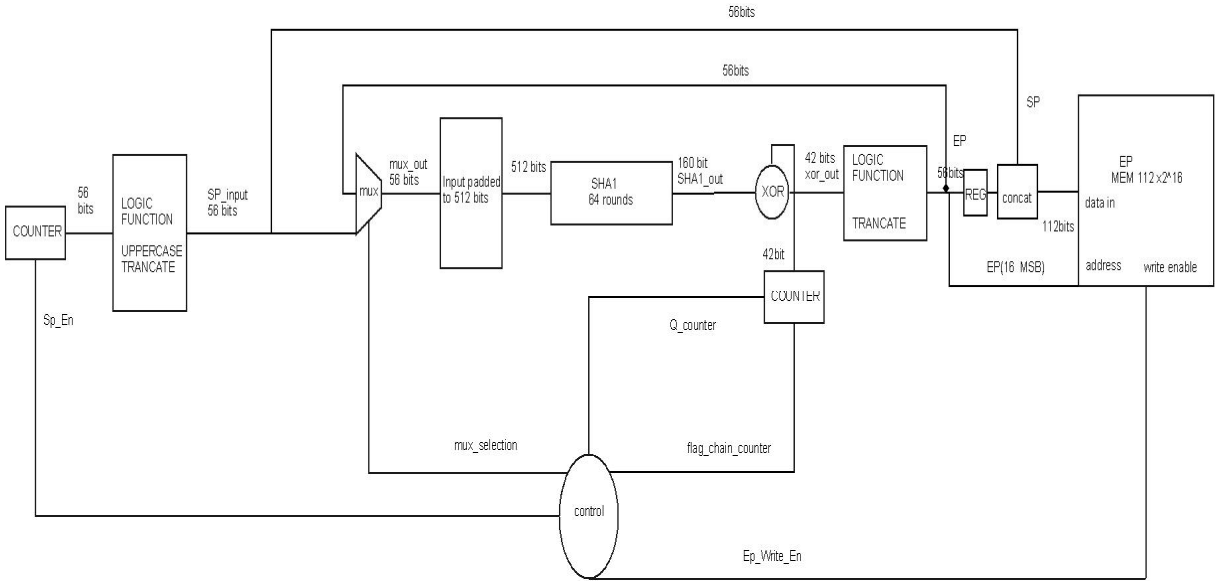
3) Το Ep_store που αποτελείται από την μνήμη και τον δείκτη διευθυνσιοδότησης της μνήμης

4) Ένα controller που ελέγχει τα σήματα για την εγγραφή στις μνήμες , και την έξοδο του πολυπλέκτη του σταδίου της παραγωγής αλυσίδας . Ουσιαστικά είναι μια FSM η οποία εξασφαλίζει και το χρονισμό των σημάτων εγγραφής στην μνήμη.

Για τον MD5 και τον SHA1 αρχικά χρησιμοποιήσαμε πλάτος κλειδιού 512 bits με δύο μνήμες και εδώ. Ενώ η συνάρτηση g , κάνει με 384 bits (512/8 bits = 64 χαρακτήρες επί 6 bits Πληροφορία = 384) .



Σχήμα 5.3 Rainbow tables για τον MD5



Σχήμα 5.4 Rainbow tables για τον SHA-1

Εδώ πρέπει να αναφέρουμε ότι γενικά οι μνήμες που χρησιμοποιήσαμε ήταν οι μεγαλύτερες δυνατές που μπορούσαν να παραχθούν για να χωρέσουν σε μια FPGA VIRTEX5 device XC5VLX330T , είναι 648x18 kb = 11,664 kb -> (11,664*1024) = 11943936 bits -> 11943936 bits / 112 bits = 106.642 , Συνεπώς επιλέγουμε βάθος 2^16 = 65536 bits. Την παραγωγή μνήμης με coregen αλλά και το synthesize και το simulate πραγματοποιήθηκαν στον server: Parmenion λόγω ότι το bottleneck του προβλήματος μας είναι η μνήμη, σε κανονικό Pc δε μπορέσαμε να εργαστούμε λόγω μικρής μνήμης . Όμως οι πραγματικοί πίνακες ουρανίων τόξων έχουν πάνω από 3.000.000. έγραφες συνεπώς οι πίνακες που παράγουμε εμείς είναι πάρα πολύ μικροί, Αφήνεται για μελλοντική δουλειά η διαδικασία παραγωγής πολλών πινάκων και η μεταφορά τους από την FPGA στο σκληρό δίσκο από ένα PC και ο υπολογισμός του συνολικού χρόνου για τη δημιουργία μεγάλων πινάκων .

5.2.1. Βελτιστοποίηση στο πλάτος κλειδιού

Επειδή το βασική αδυναμία του Lmhash είναι ότι οι κωδικοί πρόσβασης πιο μεγάλοι από 7 χαρακτήρες διαιρούνται σε δύο κομμάτια και κάθε κομμάτι κωδικοποιείται χωριστά συνεπώς η επεξεργασία έγινε σε τμήματα των 56bits και όχι των 112.

Η πιθανότητα επιτυχίας είναι σχεδόν πάντα Μηδέν(0) με την επιλογή του g για τον MD5 και τον SHA1 όσες επαναλήψεις γίνουν σε μια αλυσίδα, γι' αυτό το λόγο προχωράμε σε παραδοχή ότι η είσοδος μας θα ναι 56 bits όπως αναλύσαμε παραπάνω, σε προηγούμενο κεφάλαιο .

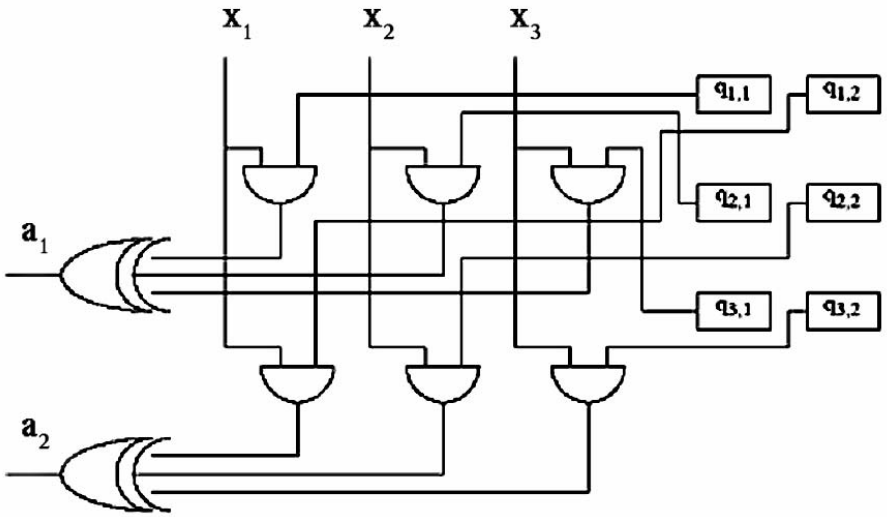
5.2.2. Βελτιστοποίηση στη μνήμη

Επόμενη βελτιστοποίηση που πραγματοποιήσαμε ήταν στις μνήμες, αντί να χρησιμοποιούμε 2 μνήμες χρησιμοποιούμε 1 και τα ζευγάρια γράφονται στη μνήμη ταξινομημένα βάση του EP. Για τον λόγο αυτό επιλέγαμε η address της μνήμης να είναι τα MSB του EP.

5.2.3. Βελτιστοποίηση στην λειτουργικότητα των rainbow tables

1) Αλλάξαμε τον αρχικό counter ώστε να παράγει μόνο τους χαρακτήρες που είναι επιτρεπτά από τους κρυπταλγόριθμους μας . Έτσι τα SP που παράγονται δεν είναι τα ίδια για πολλούς κύκλους μιας και με απλό counter παίρνοντας από τις λογικές πύλες πολλές τιμές του counter μετασχηματίζονταν σε ίδια SP.

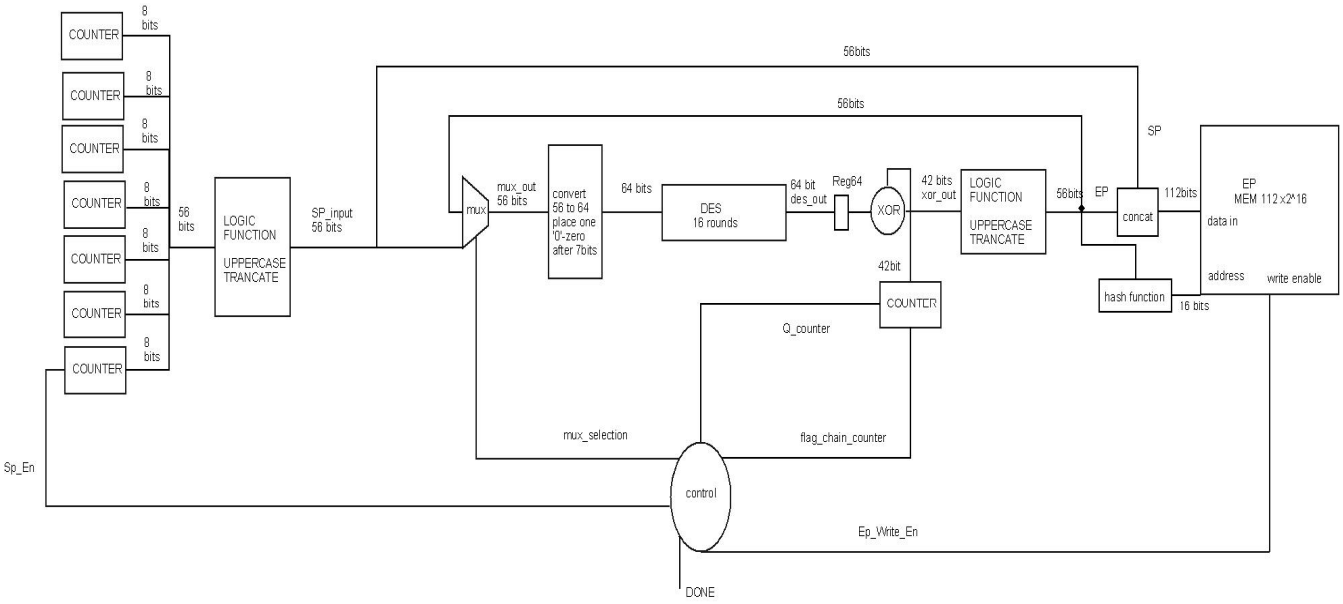
2) Επειδή , τα MSB της μνήμης δεν ήταν τα καταλληλότερα για την αντιστοίχηση 56 bits στο index της μνήμης μιας και για δε αλλάζουν όλα τα bits , χρησιμοποιήσαμε μια 3hash function όπως φαίνεται στο παρακάτω σχήμα ώστε να τα 56 bits να αντιστοιχούνται στο index της μνήμης. Τα x είναι τα bits εισόδου ,τα a είναι τα bits εξόδου και τα q είναι κάποιοι τυχαίοι πίνακες βάση των οποίων γίνεται η αντιστοίχηση.



Σχήμα 5.5 3hash function

3) Με το να γράφουμε στη μνήμη μη σειριακά , άλλα τυχαία , δημιουργείται το πρόβλημα ότι δε ξέρουμε πότε γέμισε η μνήμη , συνεπώς για παράδειγμα αν μια μνήμη έχει βάθος 1024 και εμείς παράγουμε 1024 ζευγάρια SP-EP σίγουρα κάποια από αυτά θα εγγραφούν στην ίδια θέση της μνήμης , έτσι κάποιες θέσεις της μνήμης θα μείνουν κενές . Η βέλτιστη λύση είναι να χρησιμοποιήσουμε μια CAM (context addressable memory) όμως το μέγεθος των ήδη υπάρχων CAM είναι πολύ μικρές .
Για να λύσουμε αυτό το πρόβλημα απλά βάλαμε ένα μετρητή να μετρά πως εγγραφές γίνανε στη μνήμη , με test bench παρατηρήσαμε ότι η διασπορά με την hash που χρησιμοποιήσαμε είναι γενικά καλή , αφήνει όμως κάποιες θέσεις κενές.

4) Τέλος με το που ολοκληρώνεται ο αλγόριθμος έπρεπε να διαβάζουμε τα ζευγάρια που έχουν γραφτεί στη μνήμη συνεπώς προσθέσαμε ένα πολλυπλέκτη ο οποίος επιλέγει μια εξωτερική address η οποία την δίνουμε από το PC για να διαβάζει τα αποτελέσματα σειριακά από την μνήμη της FPGA και να τα στέλνει πάλι στο PC τα αποτελέσματα.



Σχήμα 5.6 Rainbow tables στην τελική μορφή για τον Lmhash

5.2.4. Παραλληλισμός

Κάνοντας synthesize στη XILINX παρατηρήσαμε ότι η σχεδίαση μας καταλαμβάνει μόλις το 1% των πόρων της FPGA συνεπώς προχωρήσαμε στο παραλληλισμό

Υλοποιήσαμε για 2 ,4 , 8, 16, 32, 64, 128 παράλληλα μηχανάκια στην FPGA και στα 128 περάσαμε το 100% , συνεπώς οι πίνακες μας γεμίζουν σχεδόν 64 φορές πιο γρήγορα από την αρχική μας υλοποίηση μιας ο κύκλος ρολογιού ελάχιστα άλλαξε.

Οι αλλαγές που πραγματοποιήσαμε κατά τον παραλληλισμό :

Ανάλογα πόσα tables βάλαμε να γίνονται παράλληλα τόσο μειώναμε το βάθος της μνήμης , και αυξάναμε τον αριθμό των μνημών

Παράλληλα στα αρχικά SP αλλάξαμε τη σειρά των counter για κάθε table που παράγεται ώστε να έχουμε σχεδόν τυχαία και διαφορετικά SP για κάθε table

5.3 Αποτελέσματα , Συγκρίσεις

Στο κεφάλαιο αυτό παρουσιάζουμε όλα τα αποτελέσματα που είχαμε και τα συγκρίνουμε με την υλοποίηση των tables σε software.

5.3.1 Αποτελέσματα από το synthesize

Παραθέτουμε παρακάτω ένα πίνακα με τη μέγιστη συχνότητα και το ελάχιστο χρόνο για κάθε σχεδίαση που αναφέραμε παραπάνω . Παρατηρούμε ότι υπάρχει μία βελτίωση στη συχνότητα και στη περίοδο .

	Minimum period	Maximum Frequency
LM hash	5.200ns	192.308MHz
MD5	12.095ns	82.680MHz
SHA-1	7.890ns	126.744MHz

Πίνακας 5.1 Με 2 μνήμες για μια μηχανή στο H/W

	Minimum period	Maximum Frequency
LM hash	5.118ns	195.398MHz
MD5	12.091ns	82.706MHz
SHA-1	7.890ns	126.744MHz

Πίνακας 5. 2 Με 1 μνήμη για μια μηχανή στο H/W

	Minimum period	Maximum Frequency
LM hash	5.342 ns	187.192MHz
MD5	12.575ns	79.506MHz
SHA-1	8.320ns	120.432MHz

Πίνακας 5.3. Με 1 μνήμη για 32 μηχανές στο H/W

	Minimum period	Maximum Frequency
LM hash	5.508ns	181.543MHz
MD5	13.261ns	75.406MHz
SHA-1	8.715ns	114.744MHz

Πίνακας 5.4. Με 1 μνήμη για 64 μηχανές στο H/W

	Number of Slice LUTs	Number of Block RAM/FIFO
LM hash	1%	69%
MD5	1%	69%
SHA-1	1%	69%

Πινακας 5.5 Slice Logic Utilization για 1 μηχανη

	Number of Slice LUTs	Number of Block RAM/FIFO
LM hash	70%	69%
MD5	65%	69%
SHA-1	67%	69%

Πινακας 5.6 Slice Logic Utilization για 64 μηχανές

5.3.2 Υπολογισμός Χρόνου δημιουργίας Πινάκων ουρανίου τόξου

Ο υπολογισμός για το πόσο χρόνο κάνει να συμπληρωθεί ένας πίνακας εξαρτάται από το χρόνο της ελάχιστης περιόδου , μιας και οι υπολογισμοί είναι συγκεκριμένοι. Παραθέτουμε 2 πίνακες υπολογισμού των πινάκων με μεγάλο success rate, οι χρόνοι αυτοί αφορούν την υλοποίηση σε H/W με μια μηχανή

m	N	t	P rainbow = 1-(1-m/N)^t	Κύκλοι μιας επανάληψης	Ελάχιστη περίοδος ρολογιού (ns)
131072	2^42	66,000,000	0.86	17	5.2
16384	2^42	520,000,000	0.86	17	5.2
32768	2^42	265,000,000	0.86	17	5.2
65536	2^42	100,000,000	0.86	17	5.2
65536	2^42	30,000,000	0.36	17	5.2

Πίνακας 5.7. Υπολογισμός του τελικού χρόνου για τη δημιουργία tables *για μια μηχανή στο H/W*

Κύκλοι μιας επανάληψης(ns)	Χρόνος υπολογισμού αλυσίδας(ns)	Χρόνος υπολογισμού πίνακα (ns)	Χρόνος υπολογισμού πίνακα(days)
88.4	5834400000	764726476800000	9
88.4	45968000000	753139712000000	9
88.4	23426000000	767623168000000	9
88.4	8840000000	579338240000000	7
88.4	2652000000	173801472000000	2

Πίνακας 5.8. Υπολογισμός του τελικού χρόνου για τη δημιουργία tables *για μια μηχανή στο H/W*

5.3.3. Σύγκριση με Software

Συγκρίναμε τα αποτελέσματα του χρόνου υπολογισμού των πινάκων σε H/W με το χρόνο υπολογισμού σε S/W βάση του προγράμματος rainbow_crack που περιέχει τη συνάρτηση rtgen(generate rainbow table) και υπολογίζει το χρόνο σε software.

t	P rainbow = 1-(1-m/N)^t	Κύκλοι μιας επανάληψης	Ελάχιστη περίοδος ρολογιού (ns)	Κύκλοι μιας επανάληψης(ns)
5	7.45058E-08	17	5.2	88.4
1000	0.087486807	17	5.2	88.4
10000	0.141516563	17	5.2	88.4
20000	0.533706241	17	5.2	88.4
100000	0.599694531	17	5.2	88.4

Πίνακας 5.9 υπολογισμού πιθανότητας επιτυχίας πινάκων rainbow tables

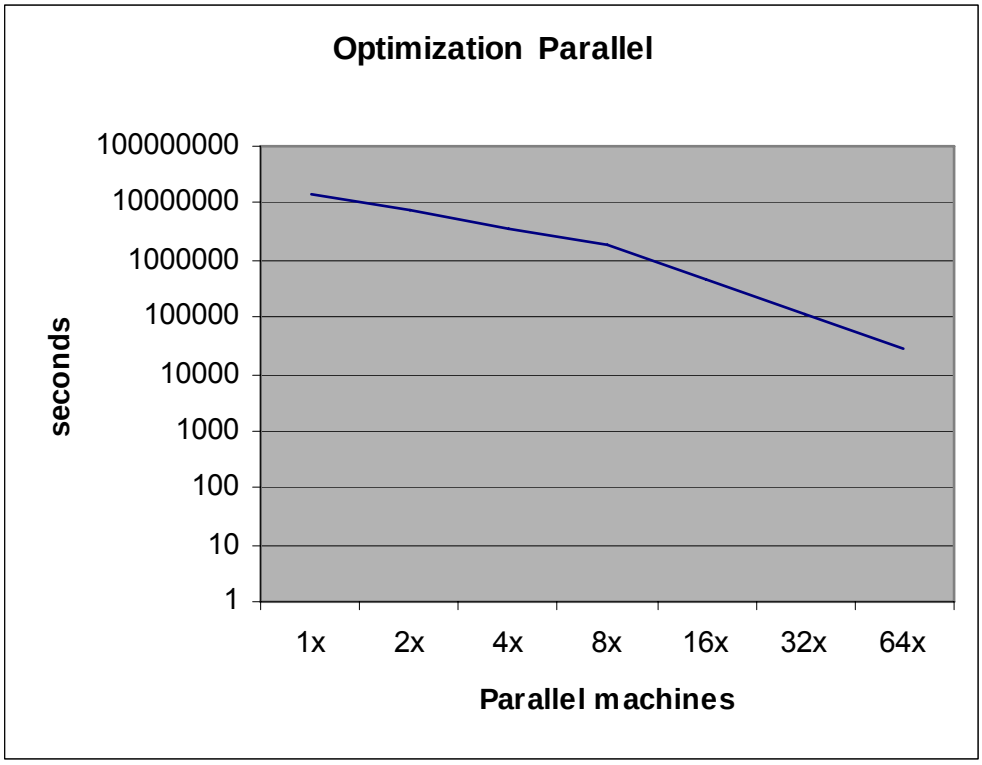
Χρόνος υπολογισμού αλυσίδας(ns)	Χρόνος υπολογισμού πίνακα (ns)	Χρόνος υπολογισμού πίνακα(sec) H/W	Χρόνος υπολογισμού πίνακα(sec) S/W	Απόδοση (φορές γρηγορότερο)
442	28,966,912	0	1	35
88,400	5,793,382,400	6	223	38
884,000	57,933,824,000	58	1565	27
1,768,000	115,867,648,000	116	5579	48
8,840,000	579,338,240,000	579	11239	19

Πίνακας 5.10 Σύγκριση των αποτελεσμάτων του παραπάνω πίνακα με τα ίδια δεδομένα σε software καθώς και η απόδοση για 1 μηχανή

Τα test bench που πραγματοποιήσαμε αφορούν μικρές επαναλήψεις (t) για τη δημιουργία μιας αλυσίδας γιατί για να συγκρίνουμε τις καταστάσεις όπου έχουμε και μεγάλο Success rate χρειάζονται πολλές μέρες για να ολοκληρωθούν τα test bench στο pc .

5.3.4 Χρόνοι με παραλληλισμό

Ο επόμενος πίνακας μας δείχνει τη μεταβολή του χρόνου όσο αυξάνεται ο παραλληλισμός. Παρατηρούμε ότι με 64 παράλληλα έχουμε αρκετή μείωση του χρόνου. Ανάλογη μείωση έχουμε και στο Pc αλλά είναι ακόμα πιο γρήγορη η ολοκλήρωση των rainbow tables στην FPGA.



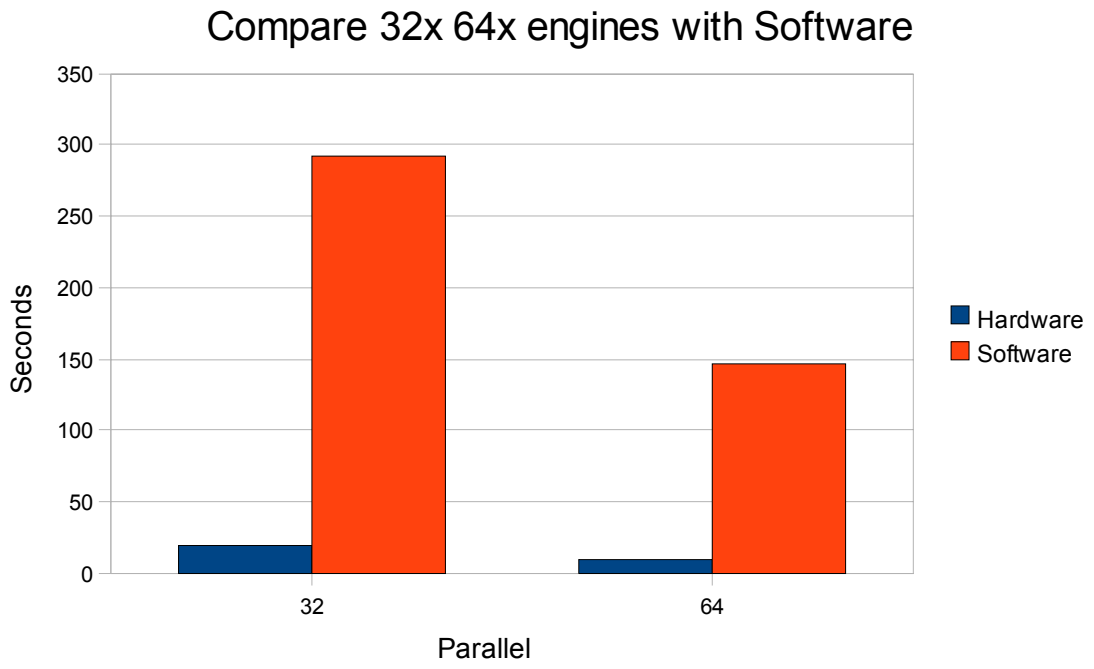
Σχήμα 5.7 Παραλληλισμός και μεταβολή στο χρόνο παραγωγής πινάκων

Το επόμενο σχήμα μας δείχνει την απόδοση(speed up) για 32 μηχανές και 64 , σε σύγκριση με το Software. Ουσιαστικά αυτό που συγκρίνουμε είναι το εξής , το χρόνο που κάνει να παράγει 1 πίνακα σε software μια μηχανή βάθους 1/64 και 1/32 του συνολικού βάθους που παράγει το hardware.

Συγκρίναμε μια παραγωγή πίνακα με στοιχεία

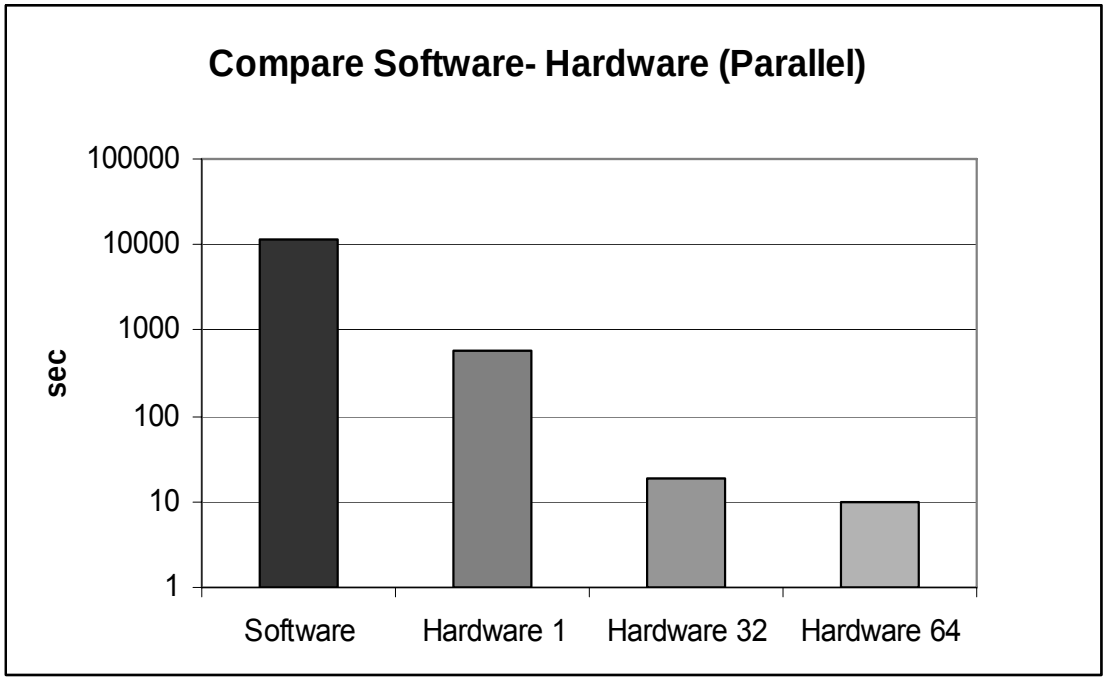
Power 2 tou m (bits)	N	P rainbow = 1-(1-m/N)^t	t	Περιοδος ρολογιου	Κύκλος μιας επανάληψης	Χρονος υπολογισμου αλυσιδας	Χρονος υπολογισμου πινακα	sec
65536	2^42	0,00	100.000	5,51	93,64	9363600	9588326400	10
65536	2^42	0,00	100.000	5,34	90,81	9081400	18598707200	18,6

Πίνακας 5.11 Αποτελέσματα παραδείγματος για Hardware για 32 και 64 μηχανές



Σχήμα 5.8. Σύγκριση 32 και 64 παράλληλων μηχανών με το software για μια μηχανή αλλά με βάθος 1/64 και 1/32 της μνήμης που παράγει σε Hardware

Αν συγκρίνουμε 1 μηχανάκι σε Software με 32 και 64 μηχανάκια σε Hardware για την παραγωγή ιδίου βάθους μνήμης τότε τα αποτελέσματα για τους παραπάνω πίνακες είναι πιο θεαματικά από ότι στο παραπάνω σχήμα



Σχήμα 5.9. Σύγκριση Software – Hardware 1 , 32 και 64 παράλληλων μηχανών με το software για μια μηχανή

6. **ΜΕΛΛΟΝΤΙΚΗ ΔΟΥΛΕΙΑ**

Η παρούσα εργασία επιδέχεται βελτιώσεις βάση των συμπερασμάτων που παρουσιάζονται από το Oechslin False Alarm Detection Using Checkpoints. Επειδή μεγάλο ποσό της δουλειάς ξοδεύεται κατά την διαδικασία της κρυπτανάλυσης εξαιτίας των λεγόμενων «false alarms» ο Oechslin παρουσίασε μια μέθοδο ανίχνευσης αυτών η οποία μειώνει σημαντικά το χρόνο κρυπτανάλυσης χρησιμοποιώντας ένα ασήμαντο ποσοστό μνήμης. Η μέθοδος αυτή βασίζεται στα check points και μειώνει το χρόνο κατά πολύ περισσότερο μιας και μείωση της μνήμης σημαίνει αύξηση στην απόδοση. Αν και ξεκινήσαμε την υλοποίηση αυτών των βελτιστοποιήσεων στη πορεία το διακόψαμε λόγω ότι μπορεί μεν να αποτελεί καινούργια προσέγγιση η οποία μπορεί να οδηγήσει σε ενδιαφέροντα αποτελέσματα όμως όσον αφορά τη σχεδίαση μας δημιουργούσε απορίες που ήταν προς το παρόν αδύνατο να επιλυθούν.

Επίσης αξιοποιώντας τα αποτελέσματα της εργασίας του **Biryukov [βιβλ. 1]** μπορούν να αποφέρουν περισσότερες βελτιώσεις στην απόδοση της υλοποίησης μας.

Επιπλέον , για να είναι πιο σωστά τα αποτελέσματα μας , και η σύγκριση με το s/w χρειάζεται οι πίνακες που παράγουμε, μιας και είναι μικροί σε βάθος να μεταφερθούν στο δίσκο η σε εξωτερική μνήμη της FPGA όπου και θα ταξινομηθούν μεταξύ τους και να μετρηθεί ο συνολικός χρόνος δημιουργίας table . Εκτίμηση είναι ότι αυτός ο χρόνος θα είναι λιγότερος μιας και το χρονοβόρο κομμάτι δεν είναι το διάβασμα της μνήμης αλλά η εκτέλεση του αλγορίθμου.

Στους χρόνους που παραθέσαμε δε προσθέσαμε τους χρόνους μεταφοράς των πινάκων από την FPGA στο PC , για την περίπτωση της μεταφοράς και της επεξεργασίας των πινάκων στο δίσκο. Συνεπώς χρειάζεται να ερευνηθεί και προσθέσει στο συνολικό χρόνο αυτή τη καθυστέρηση που όμως εκτιμάται ότι δε θα μεταβάλλει τα αποτελέσματα,

Αφού ταξινομηθούν οι εισαγωγές στους πίνακες χρειάζεται να υλοποιηθεί η διαδικασία σπασίματος του κωδικού, δηλαδή να ανατρέχει στις τιμές της εξόδου της

FPGA μέχρι το κλειδί να βρεθεί. Αυτό το κομμάτι μπορεί να γίνει στο software η και πιο γρήγορα σε FPGA , μια Pipeline αναζήτηση μπορεί να γίνει σε λίγα λεπτά.

Η υλοποίηση μας έγινε για VIRTEX5 όμως στο τέλος το κατεβάσαμε σε VIRTEX2PRO την μόνη FPGA που διαθέταμε στο εργαστήριο , αποτελεί λοιπόν μελλοντική δουλεία το κατέβασμα στη FPGA που πραγματοποιήσαμε το simulate.

Επιπλέον το πρόβλημα της διασποράς εγγραφής στη μνήμη ταξινομημένα βάση του ΕΡ το λύσαμε με την 3hash όμως παρόλα αυτά χρειάζεται περαιτέρω ψάξιμο τρόπου για καλύτερη διαχείριση αυτού του προβλήματος .

Το πρόβλημα του σπασίματος αλυσίδας δε το εξετάσαμε μιας και η υλοποίηση μας αφορούσε 1 πίνακα με βέλτιστο success rate μικρό σε μέγεθος λόγω των δυνατοτήτων που μας έδινε η FPGA (που βάση του Oechslin δεν υπάρχει θεωρητικά merge), είναι αναγκαίο στη συνέχιση του προβλήματος ο έλεγχος για merge κατά την διάρκεια της εγγραφής των μικρών τμημάτων σε 1 μεγάλο table.

Τέλος επειδή η εξέλιξη της τεχνολογίας γίνεται με γοργούς ρυθμούς , είναι αναγκαίο ο έλεγχος του όλου προβλήματος σε νέες εκδόσεις των FPGA μιας και αδυναμίες που αντιμετωπίσαμε (π.χ μικρές μνήμες) σε μελλοντικές συσκευές να λυθούν .

7. ΣΥΜΠΕΡΑΣΜΑΤΑ

Σε αυτή την εργασία παρουσιάσαμε μία αρχιτεκτονική FPGA για την διάσπαση κωδικών χρησιμοποιώντας την προσέγγιση με τους πίνακες ουρανίου τόξου του Oechslin. Οι κωδικοί που είναι στόχοι τις επίθεσης αποτελούνται από κεφαλαία, μικρά γράμματα, αριθμούς και μερικούς ειδικούς χαρακτήρες, δηλ. 42bit κωδικούς για τους MD5 και SHA1 ενώ για τον Lmhash είναι επίσης 42 bit αλλά αποτελούνται από κεφαλαία και αριθμούς. Η πλατφόρμα εγκατάστασης είναι η VIRTEX5. Παρατηρήσαμε ότι τελικά η δημιουργία των tables στην FPGA είναι πολύ ταχύτερη από ότι σε PC.

8. ΒΙΒΛΙΟΓΡΑΦΙΑ

1.**Biryukov, S. Mukhopadhyay, and P. Sarkar. Improved time-memory trade-offs with multiple data.** In B. Preneel and S. Tavares, editors, Proceedings of the 12th Annual Workshop on Selected Areas in Cryptography, Lecture Notes in Computer Science, page 19 pages. Springer, 2005.

2.**Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers.** In Tatsuaki Okamoto, editor, Advances in Cryptology: Proceedings of ASIACRYPT, volume 1976 of Lecture Notes in Computer Science, pages 1–13. Springer, 2000.

3. Making a Faster Cryptanalytical Time-Memory Trade-Off**, Philippe Oechslin,** Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17–21, 2003, Proceedings. Lecture Notes in Computer Science 2729 Springer 2003, ISBN 3-540-40674-3 http://lasecwww.epfl.ch/~oechslin/publications/crypto03.pdf

4.**Password Cracking: Rainbow Tables Explained** *Philippe Oechslin, Ph.D., CISSP* (ISC)2 Newsletter, Mar-Apr 2005

5.https://www.isc2.org/cgi-bin/content.cgi?page=738

6.**Project RainbowCrack** http://www.antsight.com/zsl/rainbowcrack/

7.**OPHCRACK (the time-memory-trade-off-cracker)** http://lasecwww.epfl.ch/~oechslin/projects/ophcrack/

8.**Rainbow table From Wikipedia, the free encyclopedia** http://en.wikipedia.org/wiki/Rainbow_table

9. **Cracking Unix Passwords using FPGA Platforms(**Nele Mentens, Lejla Batina, Bart Preneel and Ingrid Verbauwhede . Katholieke Universiteit Leuven, ESAT/SCD-COSIC http://www.hyperelliptic.org/tanja/SHARCS/talks/Mentens_et_al_paper.ps

10. **Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password *Cracking*** http://www.cosic.esat.kuleuven.be/publications/article-762.pdf

11.Time-Memory Trade-Offs: False Alarm **Detection Using Checkpoints (*Gildas Avoine, Pascal Junod, and Philippe Oechslin)***
http://lasecwww.epfl.ch/~oechslin/publications/oechslin-indocrypt-05.pdf

12. **M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*,** 1980.

13.**How Rainbow Tables work** http://kestas.kuliukas.com/RainbowTables/

14.**Τεχνικές Κρυπτογραφίας & κρυπτανάλυσης** Β.Α. Κάτος – Γ.Χ Στεφανίδης

15.**Applied Cryptography**, Second Edition , Bruce Schneier

16.K. Kusuda and T. Matsumoto. **Optimization of time-memory trade-off cryptanalysis and its application to DES**, FEAL-32 and Skipjack. *IEICE Transcations on Fundamentals of Electronics, Communications and Computer Science*, E-79A:35– 48, 1996.

17.A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. ***Handbook of Applied Cryptography***. CRC Press, 1997.

18.P. Oechslin. **Les compromis temps-m˙emoire et leur utilisation pour casser les mots de passe Windows.** In *Symposium sur la S˙ecurit˙e des Technologies de l'Information et de la Communication SSTIC, Rennes*, June 2004.

19.J.-J. Quisquater, F.-X. Standaert, G. Rouvroy, and J.D. Legat. **A cryptanalytic time-memory trade-off: First FPGA implementation**. In *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications (FPL)*, volume 2438 of *Lecture Notes in Computer Science*, pages 780–789. Springer-Verlag, 2002.

20.**R. Rivest. The MD5 Message-Digest Algorithm.**
http://www.ietf.org/rfc/rfc1321.txt , 1992.

21.**Lan Manager Hash (LM hash) From Wikipedia the free encyclopedia**
http://en.wikipedia.org/wiki/LM_hash

22.**SHA hash functions From Wikipedia, the free encyclopedia**
http://en.wikipedia.org/wiki/SHA_hash_functions

23.**MD5 From Wikipedia, the free encyclopedia**
http://en.wikipedia.org/wiki/MD5

24.**Implementation of DES Algorithm Using FPGA Technology from Arnaud Lagger** www.alagger.com/des-vhdl/report.pdf

25. **A. Fiat and M. Naor. Rigorous time/space tradeoffs for inverting functions**. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 534–541, 199

26. **J. Borst, B. Preneel, and J. Vandewalle. On the memory trade-off between exhaustive key-search and table precomputation.** In *Proceedings of*

the 19th Symposium on Information Theory in the Benelux, pages 111–118. Werkgemeenschap voor Informatie- en Communicatietheorie, Enschede, The Netherlands, 1998.

27. **J.-J. Quisquater and J. Stern.** Time-memory tradeoff revisited. *Unpublished*, 1998.

28. **M. J. Wiener. The full cost of cryptanalytic attacks.** *Journal of Cryptology*, 17(2):105–124, 2004.

29. **DES From Wikipedia, the free encyclopedia**
http://en.wikipedia.org/wiki/Data_Encryption_Standard

30.**SystemC/Verilog MD5:Source Code**
http://www.opencores.org/projects.cgi/web/systemcmd5/overview

31. **SHA cores: Overview:Source Code**
http://www.opencores.org/projects.cgi/web/sha_core/overview

32. **Επίθεση Ωμής Βιας – Επιθέσεις Λεξικών**
http://www.tech-faq.com/lang/el/brute-force-attack.shtml