ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Σχολή Μηχανικών

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Lightweight security mechanisms for μικρής κλίμακας ενσωματωμένες συσκευές

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΓΕΩΡΓΙΟΣ ΔΑΟΥΤΗΣ ΤΠ4219

Επιβλέπων καθηγητής: ΓΕΩΡΓΙΟΣ ΚΟΡΝΑΡΟΣ

Ημερομηνία παρουσίασης : 1/7/2020

Abstract

The purpose of this thesis is to safely assign and execute processing tasks in an embedded system microprocessor through authentication, encryption and resource-controlling techniques, that are separate and isolated and committed to this goal. In addition, the task scheduling process has been isolated and a microcontroller has been dedicated to performing this task .

Σύνοψη

Στόχος αυτής της πτυχιακής εργασίας είναι η ασφαλής ανάθεση και εκτέλεση επεξεργαστικών εργασιών σε έναν μικροεπεξεργαστή ενσωματωμένου συστήματος, μέσω τεχνικών αυθεντικοποίησης, κρυπτογράφησης και ταυτόχρονα δέσμευσης πόρων, ξεχωριστών, απομονωμένων και δεσμευμένων για αυτόν τον στόχο. Επιπλέον, η διαδικασία χρονοδρομολόγησης εργασιών έχει και αυτή με τη σειρά της απομονωθεί και έχει αφιερωθεί/δεσμευτεί ένας ανεξάρτητος μικροελεγκτής για την εκτέλεση αυτής και μόνον της εργασίας.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Contents

Λίστα Εικόνων

1 Εισαγωγή

1.1 Περίληψη

Ο σκοπός της εργασίας είναι να αναπτυχθεί ένα σύστημα με προβλεπόμενη, ντετερμινιστική εκτέλεση εφαρμογών/επεξεργασίας πραγματικού χρόνου και να δοθεί ιδιαίτερη σημασία σε θέματα ασφαλείας των δεδομένων, σε ένα πολυπύρηνο ενσωματωμένο σύστημα . Η ιδέα είναι ότι μία οντότητα, όπως ένας επεξεργαστής ή ένα thread (διεργασία) θα αναθέτει εργασίες σε μία άλλη οντότητα που θα τα επεξεργάζεται αφού έχουν επιλεγεί από τον χρονοπρογραμματιστή (scheduler).Τα δεδομένα προς επεξεργασία θα έχουν κρυπτογραφηθεί με συμμετρική κρυπτογραφία ώστε να διασφαλίσουμε εμπιστευτικότητα μεταξύ τον επεξεργαστών.Με αυτόν τον τρόπο θα μπορούσε να αποτραπεί μία κακόβουλη οντότητα (π.χ. επεξεργαστής, thread, εξωτερικό probe) από το να παρατηρεί ή και να τροποποιεί ευαίσθητα δεδομένα που διακινούνται ή επεξεργάζονται. Για μεγαλύτερη ασφάλεια δοκιμάσαμε μια μέθοδο στην οποία το κλειδί κάθε φορά θα δημιουργείται δυναμικά με βάση κάποια πληροφορία που θα αφήνει ο πρώτος επεξεργαστής στον δεύτερο, οπότε κάθε φορά θα είναι διαφορετικό. Με αυτό τον τρόπο μειώνουμε τον κίνδυνο κρυπτανάλυσης. Επιπλέον η κάθε εργασία θα έχει προτεραιότητα για να βοηθάει τον scheduler να επιλέξει την κατάλληλη . Η τεχνική που χρησιμοποιήθηκε για την ασφάλεια των δεδομένων είναι η τεχνική Diffie–Hellman,η όποια γενικά χρησιμοποιείται για ανταλλαγή συμμετρικού κλειδιού έτσι ώστε να μιλήσουν με ασφάλεια δυο πλευρές και ο αλγόριθμος AES-ECB για συμμετρική κρυπτογράφηση . Η υλοποίηση έγινε στο zedboard το οποίο περιλαμβάνει το Zynq®-7000. Το τελευταίο βασίζεται στην αρχιτεκτονική Xilinx® SoC που ενσωματώνει dual core ARM® CortexTM-A9 processing system (PS), που θα χρησιμοποιηθούν ως επεξεργαστές για αρχικοποίηση και κρυπτογράφηση αντίστοιχα, και programmable logic (PL). Στο PL έχουμε προσθέσει έναν επιπλέον επεξεργαστή που θα κάνει scheduling και μια μνήμη.

1.2 Κίνητρο για την Διεξαγωγή της Εργασίας – Στόχοι

Η ραγδαία ανάπτυξη ενσωματωμένων συστημάτων (σε κινητές συσκευές, έξυπνα σπίτια κ.α.), και της τεχνολογίας των συστημάτων "IoT" με χρήση πολλαπλών διαφορετικών αισθητήρων, όπως και η χρήση διαφορετικών δικτύων δεδομένων που χρησιμεύουν ως γέφυρες για την διασύνδεσή τους στο τελικό δίκτυο εγκυμονούν κινδύνους και απαιτείται κρυπτογράφηση δεδομένων. Ταυτόχρονα, σήμερα, η αυτοματοποίηση σε πολλούς τομείς της βιομηχανίας, όπως η αυτοκινητοβιομηχανία στοχεύει προς την ανάπτυξη πλήρως αυτόνομων οχημάτων με χρήση αυτόνομων τεχνολογιών. Όμως οι κίνδυνοι ασφάλειας αυξάνονται μόνο και μόνο λόγω της εξάρτησης τους από τις εφαρμογές, την ανάγκη συνδεσιμότητας και των πιο πολύπλοκων ολοκληρωμένων ηλεκτρονικών εξαρτημάτων. Διάφορες τεχνικές για ασφαλή εκτέλεση εφαρμογών σε ενσωματωμένες συσκευές έχουν ερευνηθεί στο πεδίο αυτό [1],[2],[3],[4],[5]. Παλαιότερα έχουν προταθεί αρκετές μέθοδοι [6-10] για χρονοδρομολόγηση εργασιών και αποδοτική σχεδίαση ενσωματωμένων συστημάτων για εφαρμογές με υπολογιστικές απαιτήσεις.

Ο σκοπός της δική μας εργασίας είναι να δοκιμάσουμε την απόδοση αλγορίθμων κρυπτογράφησης πάνω σε ενσωματωμένα συστήματα καθώς και τεχνικών ανταλλαγής κλειδιού συμμετρικής κρυπτογραφίας και την διαχείριση καταλλήλων πόρων του συστήματος για αυτόν τον συγκεκριμένο σκοπό.

1.3 Δομή Εργασίας

- Στο κεφαλαίο 2 κάνουμε εισαγωγή στην κρυπτογραφία/αυθεντικοποίηση χρηστών και στην ακεραιότητα των δεδομένων.
- Στο κεφαλαίο 3 ασχοληθήκαμε με τον σχεδιασμό και της ιδιαιτερότητες τον ενσωματωμένων συστημάτων καθώς και κρυπτογραφία σε τέτοιου είδους συστήματα.
- Στο κεφαλαίο 4 ασχοληθήκαμε με την περιγραφή της υλοποίησης της εργασίας και την λειτουργία της.
- Στο κεφαλαίο 5 περιγράφουμε τον τρόπο που υλοποιήσαμε την εργασία .
- Στο κεφαλαίο 6 θα δοκιμάσαμε το σύστημα και πήραμε κάποιες μετρήσεις όσον αφορά την λειτουργία του συστήματος .
- Στο κεφαλαίο 7 υπάρχουν τα συμπεράσματα και αναφέρουμε κάποιες επεκτάσεις και βελτιώσεις .

2 Κρυπτογραφία

2.1 Εισαγωγή στην κρυπτογραφία

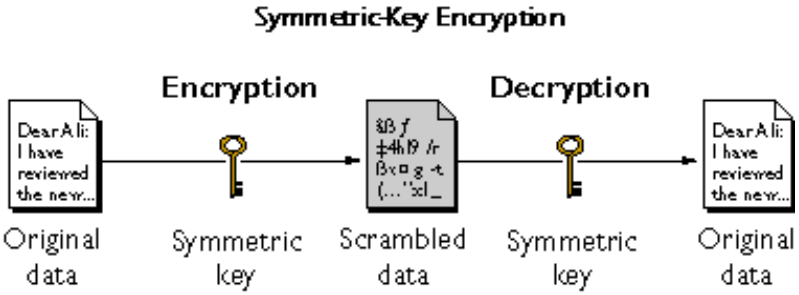
Κρυπτογραφία, είναι ένας από τους δύο κλάδους της κρυπτολογίας, και ασχολείται με τεχνικές που έχουν ως σκοπό την απόκρυψη του περιεχομένου των μηνυμάτων. Στις μέρες μας, αναφέρεται στην μελέτη και την ανάπτυξη εργαλείων και τεχνικών που ασφαλίζουν τα μηνύματα μεταξύ χρηστών ενάντια στις επιθέσεις από hackers και άλλους μη εξουσιοδοτημένους χρήστες. Η κρυπτογραφία παίζει σημαντικό ρολό όταν θέλουμε να στείλουμε μια πληροφορία μέσα από ένα μη ασφαλές δημόσιο δίκτυο, στο οποίο ο οποιοσδήποτε θα μπορούσε να καταγράψει την κίνησή του. Αυτό επιτυγχάνεται με το να μετατρέπουμε το απλό κείμενο (plaintext) σε κρυπτογραφημένο/μη κατανοητό (ciphertext) χρησιμοποιώντας ένα κλειδί και κάποιο αλγόριθμο κρυπτογράφησης που θα το χρησιμοποιήσει [11]. Αντίστοιχα η αποκρυπτογράφηση δηλαδή η μετατροπή του μη κατανοητού κείμενου στην αρχική του μορφή μπορεί να γίνει με το ίδιο κλειδί αν είναι συμμετρική κρυπτογραφία ή με το κλειδί που αντιστοιχεί στο κλειδί που κρυπτογραφήθηκε, αν πρόκειται για ασύμμετρη κρυπτογραφία.

Η βασικές λειτουργίες που παρέχει η κρυπτογραφία είναι [12] :

- ***Εμπιστευτικότητα (Privacy)***: Τα δεδομένα που πρόκειται να αποσταλούν πρέπει να είναι προσβάσιμα μόνο σε εξουσιοδοτημένα μέλη και όχι σε κάποιον τρίτο.
- ***Ακεραιότητα (Integrity)***: Διασφαλίζει ότι πληροφορία δεν μπορεί να αλλοιωθεί η να αλλαχθεί από κάποιον τρίτο αλλά μόνο από τα εξουσιοδοτημένα μέλη και δεν μπορεί να αλλοιωθεί χωρίς την ανίχνευση της αλλοίωσης.
- ***Μη απάρνηση(Non-repudiation)***: Ο αποστολέας δεν μπορεί να αρνηθεί ότι αυτός έστειλε ένα μήνυμα η εκτέλεσε μια πράξη.
- ***Πιστοποίηση (Authentication)***: Στην πιστοποίηση οι δυο πλευρές πρέπει να μπορούν να επαληθεύσουν τις ταυτότητές τους,πιο συγκεκριμένα ότι ο παραλήπτης η ο αποστολέας είναι αυτός που ισχυρίζεται και ότι οι ταυτότητές τους δεν είναι πλαστές.

2.2 Συμμετρική κρυπτογραφία

Πρόκειται για κατηγορία αλγορίθμων που χρησιμοποιούνται για να μετατρέπουν το plainx σε ciphertext και το χαρακτηριστικό τους είναι ότι χρησιμοποιούν το ίδιο κλειδί για κρυπτογράφηση και αποκρυπτογράφηση εκτελώντας κάθε φορά την αντιστροφή διαδικασία. Αν δηλαδή χρησιμοποιείται πολλαπλασιασμός για κρυπτογράφηση, για αποκρυπτογράφηση θα χρησιμοποιηθεί διαίρεση[13].



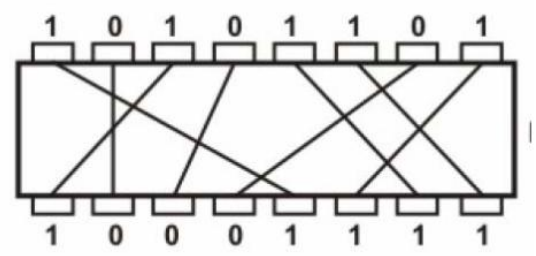
Εικόνα 2.1: Συμμετρική κρυπτογραφία

Το θετικό των συμμετρικών αλγορίθμων είναι ότι δεν απαιτούν πολλούς πόρους και εκτελούνται γρήγορα. Έχουν όμως τα εξής αρνητικά: χρειάζονται, καταρχάς, πολλά κλειδιά για να επικοινωνήσουν πολλοί χρήστες μεταξύ τους. Για παράδειγμα, αν έχουμε n χρήστες θα χρειαστούμε n(n-1)/2 διαφορετικά κλειδιά για να μπορούν όλοι μεταξύ τους να επικοινωνούν με ασφάλεια. Κατά δεύτερον, είναι δύσκολο δύο χρήστες να ανταλλάξουν ένα συμμετρικό κλειδί επειδή το κλειδί πρέπει να μεταδοθεί μέσω ενός καναλιού που μπορεί να μην είναι έμπιστο[14].

2.2.1 Τμηματικοί Κώδικες

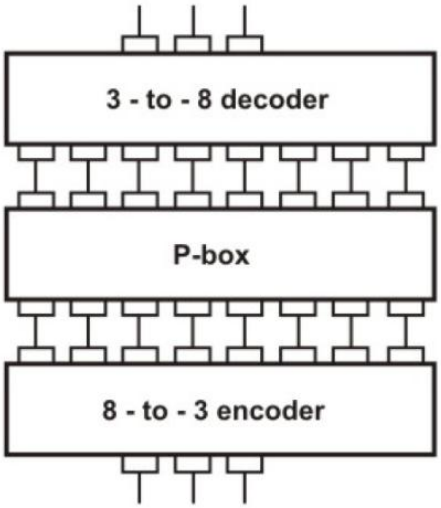
Οι τμηματικοί κώδικες [15] χωρίζουν το κείμενο σε τμήματα. Το κάθε τμήμα έχει σταθερό μέγεθος. Σε περίπτωση που δεν συμπληρώνεται ένα τμήμα, πρέπει αναγκαστικά να συμπληρωθεί για να κρυπτογραφήσει. Αν θέλουμε να κρυπτογραφήσουμε ένα τμήμα που έχει 64-bit πρέπει να εισαγάγουμε στον αλγόριθμο το τμήμα που θέλουμε να κρυπτογραφήσουμε με τιμή από 0 μέχρι 2^{64} και το αποτέλεσμα πρέπει να είναι επίσης μια τιμή από 0 μέχρι 2^{64}, ενώ η αντιστοίχιση πρέπει να είναι ένα προς ένα. Οι τμηματικοί κώδικες βασίζονται σε δυο λειτουργίες για να κρυπτογραφήσουν την πληροφορία σύμφωνα με τον Shannon η μια είναι η αντικατάσταση και η άλλη είναι η αντιμετάθεση.

Αντιμετάθεση (P-Boxes): Η αντιμετάθεση όπως φαίνεται παρακάτω διατηρεί τους αριθμούς 0 και 1, αλλάζοντάς τους απλώς σειρά.



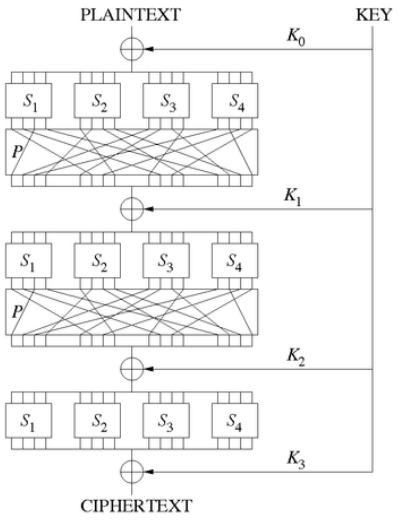
Εικόνα 2.2: Μπλοκ αντιμετάθεσης

Αντικατάσταση(S-Boxes): Η δουλειά των S-BOXES είναι να συγχέουν τα bits της εισόδου χρησιμοποιώντας ένα encoder, ένα decoder και ένα P-BOX. Η δουλειά του decoder η είναι να παίρνει σαν είσοδο N bits και σαν έξοδος 2^N bits που περιλαμβάνει μόνο ένα 1. Δηλαδή η έξοδος θα είναι της μορφής 00010000. Μετά τα δεδομένα περνάνε από ένα P-BOX και το αποτέλεσμα είναι να αλλάξει θέση το 1 και τέλος ξαναγίνεται από 2^N bits σε N bits με τον encoder.



Εικόνα 2.3: Μπλοκ αντικατάστασης

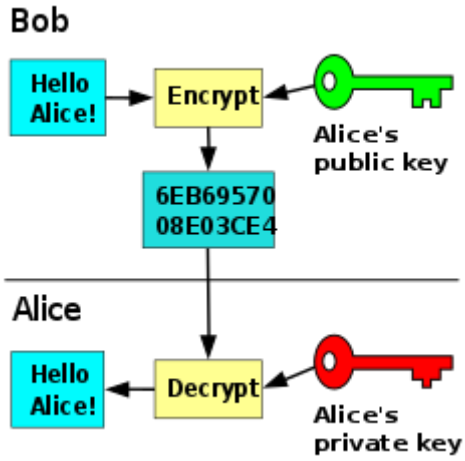
Η παραπάνω διαδικασία επαναλαμβάνεται μέχρι κάποιον αριθμό, ανάλογα με τον αλγόριθμο και το κλειδί. Όσο μεγαλύτερος είναι τόσο αυξάνεται η ασφάλεια και η πιθανότητα κρυπτανάλυσης μειώνεται. Στην παρακάτω εικόνα 2.4 βλέπουμε ένα SP-NETWORK που αποτελείται από S και P-BOXES όπου η είσοδος διαιρείται σε 8 bit κομμάτια που περνάνε από ένα S και μετά από ένα P-BOX επαναληπτικά.



Εικόνα 2.4: Αλγόριθμος κρυπτογράφησης

2.3 Ασύμμετρη κρυπτογραφία

Στην κρυπτογραφία δημόσιου κλειδιού, στην ασύμμετρη κρυπτογραφία, υπάρχουν δύο κλειδιά: ένα ιδιωτικό κλειδί και ένα δημόσιο κλειδί[16]. Το δημόσιο κλειδί είναι γνωστό σε όλους ενώ το ιδιωτικό κλειδί είναι γνωστό μόνο στον ιδιοκτήτη του αντιστοίχου δημοσίου κλειδιού. Ο αποστολέας χρησιμοποιεί το δημόσιο κλειδί του παραλήπτη για κρυπτογράφηση και ο παραλήπτης χρησιμοποιεί το ιδιωτικό του κλειδί για αποκρυπτογράφηση, όπως φαίνεται παρακάτω στο σχήμα. Δηλαδή ό,τι κρυπτογραφείται με το ιδιωτικό κλειδί αποκρυπτογραφείται με το δημόσιο και ό,τι κρυπτογραφείται με το δημόσιο κλειδί αποκρυπτογραφείται με το ιδιωτικό.



Εικόνα 2.8: Ασύμμετρη κρυπτογραφία

Τα πλεονεκτήματα είναι ότι το δημόσιο κλειδί μπορεί να χρησιμοποιηθεί από οποιοδήποτε για να επικοινωνήσει με αυτόν που θέλει χωρίς να χρειάζεται να ανταλλάξουν session κλειδί πάνω από ένα μη ασφαλές κανάλι και ότι σε αντίθεση με την συμμετρική κρυπτογραφία, ο αριθμός των απαιτούμενων κλειδιών είναι μικρός για επικοινωνία μεταξύ πολλών χρηστών. Τα μειονεκτήματα είναι ότι δεν είναι αποτελεσματικό για μεγάλα μηνύματα επειδή απαιτούν πολλή υπολογιστική ισχύ και ότι πρέπει να επαληθευτεί η σχέση μεταξύ ενός παραλήπτη και του δημόσιου κλειδιού έτσι ώστε να διαπιστωθεί ότι όντως του ανήκει.

Ένας από τους πιο διάσημους αλγόριθμους ασύμμετρης κρυπτογράφησης είναι ο RSA. Ο συγκεκριμένος αλγόριθμος βασίζεται στην θεωρία των αριθμών για να δημιουργήσει ένα ζευγάρι κλειδιών. Αν επιλέξουμε ένα μικρό κλειδί τότε θα έχουμε ταχύτητα, ενώ αν επιλέξουμε ένα μεγάλο, ασφάλεια. Ένα ζευγάρι κλειδιών αποτελείται από το δημόσιο (e,n) και το ιδιωτικό (d,n).

Αλγόριθμος παραγωγής για ζευγάρι κλειδιών

1)Διάλεξε p, q p και q είναι πρώτοι αριθμοί

2)Υπολόγισε το n = p x q

3)Υπολόγισε το $\Phi(n) = (p - 1)(q - 1)$

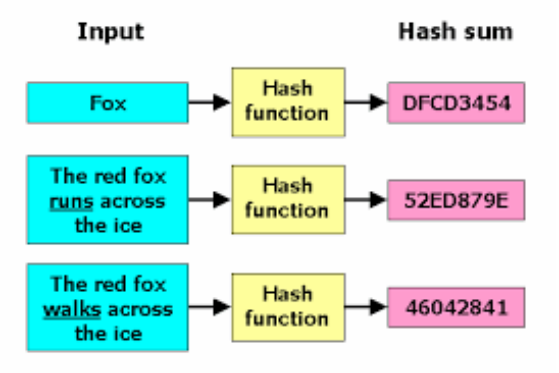
4)Διάλεξε ακέραιο e ώστε $\gcd(\Phi(n), e)=1$; $1<e<\Phi(n)$

5)Υπολόγισε το d όπου $d = e^{-1} \bmod \Phi(n)$

Δημόσιο κλειδί KU = {e,n} Ιδιωτικό κλειδί KR = {d,n}

2.4 Συναρτήσεις Κατακερματισμού

Οι συναρτήσεις κατακερματισμού είναι ένα από τα πιο σημαντικά εργαλεία στην σύγχρονη κρυπτογραφία. Χρησιμοποιούνται κυρίως για αυθεντικοποίηση χρηστών, ψηφιακές υπογραφές, για γεννήτριες τυχαίων αριθμών[17]. Και η προδιάγραφες που πρέπει να πληρούν είναι ότι ως είσοδο μπορούν να πάρουν οποιοδήποτε μέγεθος πληροφορίας και το μέγεθος του αποτελέσματος της, δηλαδή η έξοδος, πρέπει να έχει πάντα σταθερό μέγεθος, να είναι εύκολο να υπολογιστεί αλλά η αντίστροφή λειτουργία να είναι υπολογιστικά αδύνατον να βρεθεί, αρά αν έχουμε μια είσοδο X και μια συνάρτηση κατακερματισμού H να είναι εύκολο να υπολογίσουμε το H(X) αλλά αδύνατο να υπολογίσουμε το X από το H(X)



Εικόνα 2.9: Συναρτήσεις κατακερματισμού

2.5 Υβριδική Κρυπτογράφηση

Ο σκοπός της υβριδικής κρυπτογράφησης είναι να δημιουργήσει μια μέθοδος η οποία θα έχει τα πλεονεκτήματα της συμμετρικής και ασύμμετρης κρυπτογραφίας. Η συμμετρικοί αλγόριθμοι είναι γρήγοροι αλλά το αρνητικό τους είναι η διαχείριση των κλειδιών ,είναι δύσκολο να στείλουμε ένα συμμετρικό κλειδί πάνω από ένα μη έμπιστο κανάλι. Αντιθέτως η ασύμμετρη κρυπτογραφία δεν έχει πρόβλημα με το να στέλνει κλειδιά επειδή το μυστικό μέρος είναι πάντα κρυφό και αδύνατο να βρεθεί αλλά είναι υπολογιστικά πολύ απαιτητικοί και χρονοβόροι. Στην υβριδική κρυπτογράφηση ο αποστολέας μπορεί να δημιουργήσει ένα τυχαίο συμμετρικό κλειδί, να κρυπτογραφήσει την πληροφορία με το κλειδί και στο τέλος του μηνύματος να υπάρχει το κλειδί που χρησιμοποιήθηκε κρυπτογραφημένο με το δημόσιο κλειδί του παραλήπτη. Έτσι, μόνο ο παραλήπτης θα μπορεί να βρει το κλειδί και να αποκρυπτογραφήσει το μήνυμα.

2.6 Ανταλλαγή Συμμετρικών Κλειδιών με δημοσιά κρυπτογραφία

Εδώ χρησιμοποιείτε το δημόσιο κλειδί του παραλήπτη μόνο για να κρυπτογραφήσει ένα session key, το οποίο θα χρησιμοποιηθεί μόνο μια φορά σε ένα session. Ο παραλήπτης θα το αποκρυπτογραφήσει με το κρυφό κλειδί και έτσι θα έχουν ένα συμμετρικό κλειδί για να ανταλλάξουν πληροφορίες.

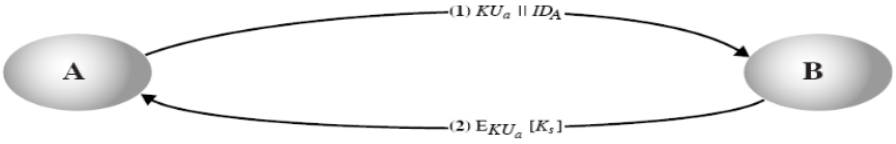
Αλγόριθμος MERKLE για ανταλλαγή συμμετρικού κλειδιού :

1) Ο αποστολέας στέλνει στον παραλήπτη το δημόσιο κλειδί και την ταυτότητά του

2) Ο παραλήπτης στέλνει ένα συμμετρικό κλειδί K για τη σύνοδο στον αποστολέα κρυπτογραφημένο με το κλειδί που του είχε στείλει πριν

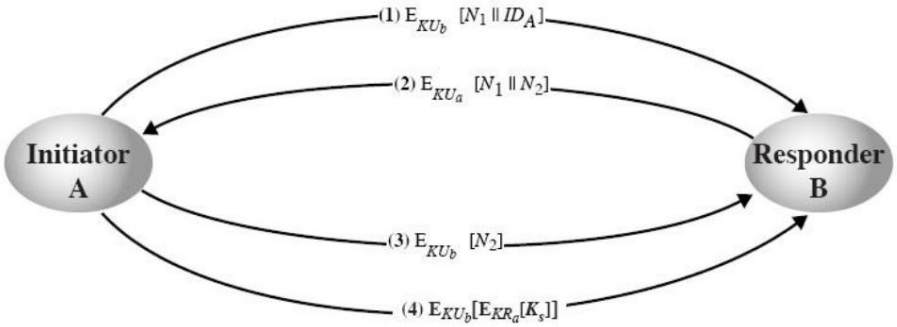
3) Ο αποστολέας αποκρυπτογραφεί το κλειδί συνόδου και το χρησιμοποιούν και οι δύο.

Το πρόβλημα με αυτό τον τρόπο είναι ευάλωτο σε man in the midle attack. Ο οποιοσδήποτε μπορεί να προσποιηθεί ότι είναι ο αποστολέας και να στείλει το δικό του κλειδί.



Εικόνα 2.10: Σχεδιάγραμμα ανταλλαγής συμμετρικού κλειδιού

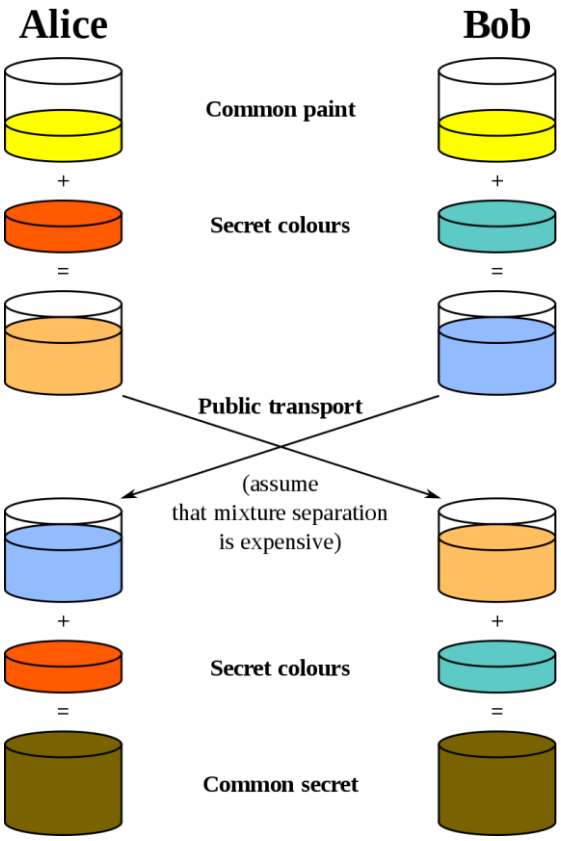
Η λύση στο πρόβλημα θα ήταν εφόσον έχουν ανταλλάξει με ασφάλεια τα δημοσιά κλειδιά, και έχουν σιγουρευτεί ότι ανήκουν σε αυτόν που θέλουν να επικοινωνήσουν. τότε ο ένας από τους 2 να κρυπτογραφήσει ένα κλειδί, πρώτα με το μυστικό κλειδί και μετά με το δημόσιο του παραλήπτη. Έτσι ο μονός που μπορεί να το αποκρυπτογραφήσει θα είναι ο αποστολέας και στην συνέχεια θα σιγουρευτεί για την προέλευση επειδή θα το αποκρυπτογραφήσει με το δημόσιο του αποστολέα.



Εικόνα 2.11: Σχεδιάγραμμα ανταλλαγής συμμετρικού κλειδιού με αυθεντικοποίηση

2.7 Ανταλλαγή Συμμετρικών Κλειδιών Diffie–Hellman

Ο αλγόριθμος Diffie–Hellman χρησιμοποιείται για την ανταλλαγή ενός κοινού μυστικού πάνω από ένα δημόσιο δίκτυο[18].Δεν πρόκειται για πρωτόκολλο που παρέχει αυθεντικοποίηση αλλά χρησιμοποιείται σε πρωτόκολλα αυθεντικοποίησης, όπως το TLS. Η γενική ιδέα φαίνεται στο παρακάτω σχήμα. Στην αρχή οι δυο πλευρές συμφωνούν σε ένα κοινό χρώμα το οποίο δεν είναι απαραίτητα κρυφό στους υπολοίπους (από κάτω είναι με κίτρινο). Μετά η κάθε πλευρά φτιάχνει ένα μυστικό χρώμα που θα χρησιμοποιήσει μόνο αυτή και θα το αναμίξει με το κοινό χρώμα. Με αυτό τον τρόπο θα προστατεύσει το μυστικό χρώμα και θα το στείλει στον άλλο. Είναι σημαντικό να είναι υπολογιστικά δύσκολο να ξεχωριστούν τα χρώματα. Τέλος, η κάθε πλευρά θα αναμίξει το χρώμα που παρέλαβε με το δικό της χρώμα καταλήγοντας εν τέλει στο ίδιο χρώμα.



Εικόνα 2.12: Σχεδιάγραμμα ανταλλαγής συμμετρικού Diffie–Hellman

Παράδειγμα με αριθμούς :

Το κοινό χρώμα είναι ένα ζευγάρι αριθμών p και q, οπού το q είναι περιττός και το p να είναι τέτοιο ώστε κάθε φορά που υψώνουμε το q σε κάποια δύναμη το υπόλοιπο της διαίρεσης, δηλαδή το q mod p, να είναι μοναδικό στο διάστημα 1 μέχρι p-1. Παράδειγμα τέτοιων αριθμών είναι και η παρακάτω εικόνα. Το 3 και το 7 ικανοποιούν τον παραπάνω περιορισμό γιατί αν υψώσουμε το 3 με όλους τους αριθμούς από το 1 μέχρι το 6 το αποτέλεσμα του modulo θα είναι μοναδικό στο διάστημα από 1 μέχρι 6.

$3^1 = 3 = 3^0 \times 3 = 1 \times 3 = 3 = 3 \pmod 7$

$3^2 = 9 = 3^1 \times 3 = 3 \times 3 = 9 = 2 \pmod 7$

$3^3 = 27 = 3^2 \times 3 = 2 \times 3 = 6 = 6 \pmod 7$

$3^4 = 81 = 3^3 \times 3 = 6 \times 3 = 18 = 4 \pmod 7$

$3^5 = 243 = 3^4 \times 3 = 4 \times 3 = 12 = 5 \pmod 7$

$3^6 = 729 = 3^5 \times 3 = 5 \times 3 = 15 = 1 \pmod 7$

$3^7 = 2187 = 3^6 \times 3 = 1 \times 3 = 3 = 3 \pmod 7$

Έστω ότι έχουμε χρηστές A και B. Αρχικά το ζευγάρι p και q μπορεί να το γνωρίζει ο οποιοσδήποτε επιλέγει το 5 και το 23 αντίστοιχα.

1) Πρώτα ο A δημιουργεί ένα κλειδί ένα τυχαίο αριθμό που τον γνωρίζει μόνο αυτός ,έστω ότι είναι το 4 και υπολογίζει το q mod p όποτε $5^4 \bmod 23 = 4$

2) Ο B επιλεγεί και αυτός έναν αριθμό για μυστικό κλειδί τον αριθμό 3, και υπολογίζει το $5^3 \bmod 23 = 10$ τώρα που το μυστικό κλειδί του καθένα είναι ασφαλείς (δηλαδή αναμιγμένο με το κοινό μέρος του κλειδιού) μπορούν να ανταλλάξουν αυτή την πληροφορία τον αριθμό που υπολόγισαν.

Για να καταλήξουν στο ίδιο μυστικό κλειδί και να μπορέσουν να επικοινωνήσουν με ασφάλεια πρέπει :

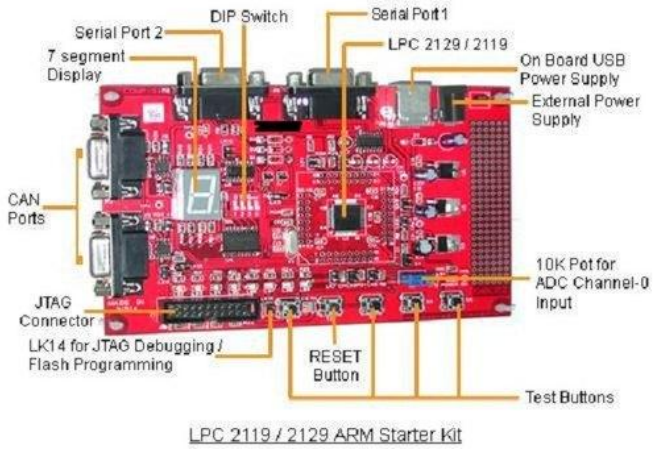
Ο A πρέπει να υψώσει στην δύναμη του μυστικού του κλειδιού αυτό που του έστειλε ο B και ο B αντίστοιχα να πάρει τον αριθμό που του έστειλε ο A και να κάνει το ίδιο (σε αυτό το σημείο ο A και ο B προσθέτουν το μυστικό χρώμα σε αυτό που παρέλαβαν για να καταλήξουν στο ίδιο μυστικό), οπότε ο A καταλήγει στο $10^4 \bmod 23 = 18$ και ο B καταλήγει στο $4^3 \bmod 23 = 18$ και μετά από αυτή την διαδικασία μπορούν να μιλήσουν με συμμετρική κρυπτογράφηση χρησιμοποιώντας τον παραπάνω αριθμό. Ο παραπάνω αλγόριθμος ισχύει γιατί ισχύει το παρακάτω:

$$A^b \bmod p = q^{ab} \bmod p = q^{ba} \bmod p = B^a \bmod p$$

3 Ενσωματωμένα συστήματα

3.1 Ενσωματωμένα συστήματα

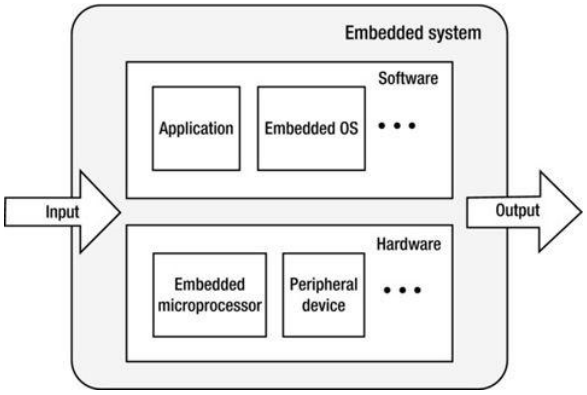
Ενσωματωμένο σύστημα είναι ένας συνδυασμός από υλικό και λογισμικό καθώς και αλλά εξαρτήματα που είναι σχεδιασμένα για μια συγκεκριμένη δουλειά [19]. Στα περιφερειακά του αυτοκινήτου, για παράδειγμα, υπάρχουν συστήματα που χειρίζονται τα φρένα, κάποια αλλά που παρακολουθούν την θερμοκρασία, τα καυσαέρια, την ταχύτητα κτλ. Όλα περιλαμβάνουν κάποιο είδος επεξεργαστή και λογισμικό για την αντίστοιχή δουλειά. Η παραπάνω δουλειές θα μπορούσαν να αντικατασταθούν από κυκλώματα ειδικού σκοπού αλλά συνήθως είναι πιο φθηνό και έχουμε περισσότερή ευελιξία αν χρησιμοποιούμε ενσωματωμένα συστήματα.



Εικόνα 3.1: Ενσωματωμένο σύστημα

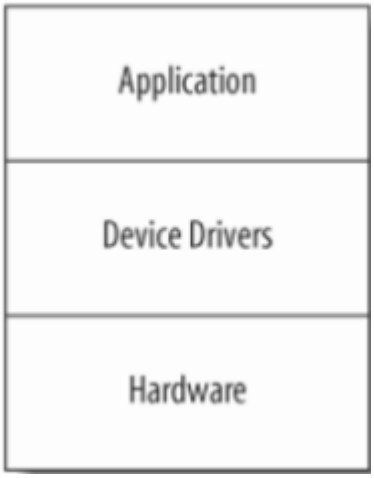
3.2 Σχεδιασμός ενσωματωμένων συστημάτων

Όλα τα ενσωματωμένα συστήματα πέρα από τον επεξεργαστή και το λογισμικό πρέπει να περιλαμβάνουν ένα μέρος για να αποθηκεύουν τον εκτελέσιμο κώδικα και ένα προσωρινό μέρος για τις μεταβλητές. Όταν θα τρέχουν τον κώδικα θα χρειαστούν read-only memory (ROM) και random access memory (RAM) αντίστοιχα. Αν δεν απαιτούν πολλή μνήμη μπορούν να αποθηκεύονται στο ίδιο chip με τον επεξεργαστή. Συνήθως όμως περιλαμβάνουν εξωτερική μνήμη. Επίσης, όλα τα ενσωματωμένα περιλαμβάνουν κάποιο είδος εισόδου και εξόδου. Για παράδειγμα, αν θέλαμε να μετρήσουμε θερμοκρασία μέσα σε ένα αυτοκίνητο θα έπρεπε το σύστημα να έχει κάποιο θερμόμετρο για είσοδο και μια οθόνη για εξόδου έτσι ώστε μπορούμε να βλέπουμε την θερμοκρασία.



Εικόνα 3.2: Γενικό σχεδιάγραμμα ενσωματωμένου συστήματος

Με εξαίρεση κάποια κοινά χαρακτηριστικά σαν την αρχιτεκτονική η λειτουργία αλλάζει από σύστημα σε σύστημα ανάλογα με την εφαρμογή που θέλουμε να κάνουμε, οπότε όπως βλέπουμε από την παρακάτω εικόνα το επίπεδο εφαρμογής (application) μπορεί να αλλάζει σε κάθε περίπτωση και είναι αυτό που αλλάζει πιο συχνά . Οι οδηγοί (drivers) περιλαμβάνουν την λειτουργία του κάθε περιφερειακού-υλικού που θέλουμε να χρησιμοποιήσουμε και ο σκοπός τους είναι να κάνουμε μια πιο φορητή εφαρμογή και να καταργήσουμε την ανάγκη για τον προγραμματιστή να γνωρίζει την λειτουργία του υλικού.



Εικόνα 3.3: Βασικά επίπεδα αφαίρεσης

3.3 Απαιτήσεις που επηρεάζουν τις επιλογές σχεδιασμού

Ανάλογα με την εφαρμογή πρέπει να έχουμε και διαφορετικές απαιτήσεις και περιορισμούς που επηρεάζουν το τελικό προϊόν.Για παράδειγμα, αν το τελικό προϊόν πρέπει να είναι φθηνό τότε πρέπει να θυσιάσουμε υπολογιστική ισχύ και την μικρή κατανάλωση ενεργείας, ενώ από την άλλη αν το σύστημα πρέπει να είναι αξιόπιστο ίσως να θυσιάσουμε το μικρό κόστος. Μερικά παραδείγματα χαρακτηριστικών που πρέπει να έχουμε υπόψη κατά τον σχεδιασμό τέτοιων συστημάτων είναι παρακάτω [20]:

Υπολογιστική ισχύ

Είναι ο φόρτος εργασίας που ο επεξεργαστής μπορεί να διαχειριστεί και ένας τρόπος να μετρηθεί είναι πόσες εκατομμύρια εντολές μπορεί να εκτελέσει μέσα σε ένα δευτερόλεπτο (millions of instructions per second) αν και δεν είναι το μόνο χαρακτηριστικό που πρέπει να εξετάζουμε αν πρόκειται για υπολογιστική ισχύ. Επίσης μετράει και το μέγεθος των καταχωρητών του επεξεργαστή που συνήθως έχει μέγεθος 8 με 16 bits σε τέτοιου είδους συστήματα.

Κατανάλωση ενεργείας

Είναι η ποσότητα ενέργειας κατά την λειτουργιά του συστήματος και είναι αρκετά σημαντικός παράγοντας αν πρόκειται για φορητές συσκευές που βασίζονται σε μπαταριά. Μια μονάδα μέτρησης κατανάλωσης ενέργειας είναι Μιλιβάτ ανά εκατομμύρια εντολές. Όσο μεγαλύτερο είναι το μέγεθος τόσο μεγαλύτερη είναι και η κατανάλωση. Πάντα η μικρή κατανάλωση είναι επιθυμητή καθώς όσο μικρότερο είναι το μέγεθος της συσκευής τόσο μικρότερη είναι και η θερμοκρασία που αναπτύσσει.

Μνήμη

Το μέγεθος της μνήμης που απαιτείται RAM και ROM που χρειάζεται για τον εκτελέσιμο κώδικα και της μεταβλητές. Η επιλογή μνήμης επηρεάζει και την επιλογή επεξεργαστή, καθώς το μέγεθος τον καταχτητών κανονίζει το μέγεθος της μνήμης που μπορεί να προσπελάσει.

Χρόνος ζωής

Πόσο καιρό το προϊόν αναμένεται να διαρκέσει ανάλογα με την χρίση του . Η αναμενόμενη διάρκεια ζωής του συστήματος επηρεάζει πολλά πράγματα, από τον σχεδιασμό μέχρι πόσο το τελικό προϊόν θα κοστίζει .

Αξιοπιστία

Η αξιοπιστία έχει να κάνει με την αποτυχία λειτουργίας του συστήματος και τις αναμενόμενες συνέπειες που θα έχει πχ. αν πρόκειται για κάποιο ηλεκτρονικό θερμόμετρο, αν χαλάσει, δεν θα υπάρχουν σημαντικές συνέπειες. Άλλα αν πρόκειται για τα φρένα του αυτοκινήτου και χαλάσουν την ώρα της οδήγησης θα υπάρχουν.

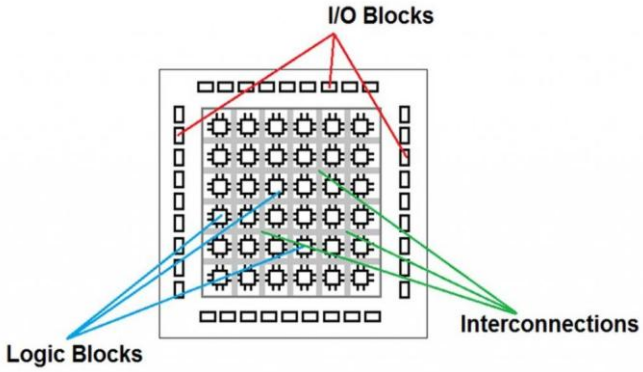
3.4 Προγραμματιζόμενη Λογική

Field-programmable gate array (FPGA) [21] η συστοιχία επιτόπια προγραμματιζόμενων πυλών είναι συσκευές ημιαγωγών που είναι βασισμένα σε ένα πίνακα από επαναπρογραμματιζόμενα λογικά στοιχειά (configurable logic blocks (CLBs), που υλοποιούν λογικές συναρτήσεις και συνδέονται μέσω προγραμματιζόμενες διασυνδέσεις (programmable interconnects).Τα FPGA μπορούν να προγραμματιστούν και μετά από την κατασκευή τους για όποια εφαρμογή θέλουμε. Αυτό τα διαφοροποιεί σε σχέση με άλλες παρόμοιες τεχνολογίες όπως Application Specific Integrated Circuits (ASICs) που είναι κατασκευασμένα για συγκεκριμένες εργασίες και δεν επαναπρογραμματίζονται.

Τα κυρία χαρακτηριστικά τον FPGA είναι:

1. Κάθε φορά που κόβεται η παροχή του, ρεύματος ο προγραμματισμός τους χάνεται οπότε πρέπει να κρατείται κάπου.
2. Μπορούμε να αλλάζουμε το κύκλωμα σαν σε λογισμικό συνήθως προγραμματίζονται χρησιμοποιώντας γλώσσες περιγραφής υλικού.
3. Έχουμε μεγαλύτερη κατανάλωση ενεργείας σε σχέση με άλλες τεχνολογίες όπως ASIC.
4. Το chip μπορούμε να το προγραμματίσουμε όσες φορές θέλουμε.

Οπότε τα FPGA συνδυάζουν την ευκολία του προγραμματιστή και τα πλεονεκτήματα των κυκλωμάτων, είναι ιδανικά για όταν θέλουμε να αλλάζουμε την σχεδίαση του κυκλώματος και να διορθώνουμε σφάλματα και για προϊόντα που παράγονται σε μικρές ποσότητες. Από την άλλη, τα ASIC είναι φτηνότερα για προϊόντα που κατασκευάζονται μαζικά σε μεγάλες ποσότητες.



Εικόνα 3.4: Εσωτερικό FPGA

Μερικές εφαρμογές FPGA είναι :

- αναγνώριση φωνής

- κρυπτογραφία
- βίο-πληροφορική
- ψηφιακή επεξεργασία σήματος

3.5 Κρυπτογραφία και ασφάλεια σε ενσωματωμένα συστήματα

Όπως αναφέραμε και σε προηγούμενα κεφαλαία τα ενσωματωμένα συστήματα είναι ειδικού σκοπού συσκευές άλλα εξίσου σημαντικές όσον άφορα την ασφάλεια όπως όλα τα άλλα συστήματα. Επιθέσεις κατά καιρούς έχουν καταγραφεί και πολλές φορές με καταστροφικά αποτελέσματα τόσο σε κόστος όσο και σε ανθρώπινες ζωές. Τα συστήματα γενικού σκοπού δεν έχουν πρόβλημα να εκτελέσουν καθώς έχουν την απαραίτητη υπολογιστική ισχύ και η παροχή τάσης δεν είναι συνήθως πρόβλημα. Για να εφαρμόσουμε όμως τις ίδιες τεχνικές πάνω σε ενσωματωμένα συστήματα δεν είναι το ίδιο εύκολο. Μερικοί λόγοι, για τους οποίους πολλές φορές αμελείται η ασφάλεια σε τέτοιου είδους συστήματα, είναι το γεγονός ότι αν εφαρμόσουμε τις ίδιες τεχνικές σε συστήματα με περιορισμένους πόρους θα είναι δύσκολο έως αδύνατο να ανταπεξέλθουν. Αυτό συμβαίνει καθώς δεν είναι ικανά να εκτελέσουν πολύπλοκες αριθμητικές πράξεις και αν οι κατασκευαστές εφαρμόσουν τέτοιες τεχνικές σε τέτοιου είδους συστήματα το τελικό προϊόν δεν θα είναι οικονομικό και ανταγωνιστικό ή θα έχει πολλές απαιτήσεις ως προς την κατανάλωση και την υπολογιστική ισχύ, κάτι που πολλές φορές δεν είναι επιθυμητό. Οπότε τα τελευταία χρονιά τεχνικές ελαφριάς κρυπτογράφησης έχουν αναπτυχθεί τόσο σε software όσο και σε hardware καθώς και συνδυασμοί των δυο ώστε η κρυπτογραφία να μην έχει επιπτώσεις στην επίδοση του συστήματος.

Σε υλικό (hardware) έχουν αναπτυχθεί αλγόριθμοι εξ ολοκλήρου σε FPGA ή σε κύκλωμα, τέτοιες τεχνικές κάνουν πιο γρήγορο και λιγότερο απαιτητικό σε κατανάλωση ενέργειας καθώς επίσης πετυχαίνουν παραλληλία σε κάποιες περιπτώσεις που είναι τα χαρακτηριστικά που θέλουμε για τέτοιου είδους συστήματα. Ένας ακόμα λόγος για να υλοποιήσουμε και σε άλλες περιπτώσεις πέρα από συσκευές με μικρές δυνατότητες είναι ότι είναι λιγότερο ευπαθείς σε επιθέσεις καθώς η μνήμη που χρησιμοποιείται για τους υπολογισμούς είναι προσβάσιμη μόνο από τους ειδικούς επεξεργαστές καθιστώντας την αποκομμένη από το υπόλοιπο σύστημα, σε αντίθεση με την κρυπτογραφία λογισμικού που δεν έχει τέτοιους μηχανισμούς [22].

Σε λογισμικό (software) έχουν αναπτυχθεί αλγόριθμοι που χρειάζονται μόνο επεξεργαστή γενικού σκοπού για να τρέξουν χωρίς κάποιο ειδικό υλικό οπότε είναι εύκολα φορητή σε αντίθεση με λύσεις σε harware. Αυτού του είδους αλγόριθμοι έχουν σκοπό να ελαττώσουν τις απαιτήσεις σε μνήμη και σε υπολογιστική ισχύ τις οποίες έχουν η κανονική αλγόριθμοι κρυπτογράφησης. Ένας τέτοιος αλγόριθμος είναι ο TEA που έχει σχεδιαστεί να είναι απλός [23] χρησιμοποιεί κυρίως XOR, ADD και μετατοπίσεις για αντικατάσταση και αντιμετάθεση που είναι υποχρεωτική για τέτοιου είδους κρυπτογραφία χωρίς να είναι ανάγκη να χρησιμοποιεί P-boxes και S-boxes που είναι απαιτητικά σε υπολογιστική ισχύ.

Η υβριδική συνδυάζει τα θετικά από τους 2 παραπάνω μεθόδους. Συνήθως οι πολύπλοκες πράξεις υλοποιούνται co-processors που είναι υλοποιημένοι σε υλικό . Το λογισμικό αναλαμβάνει να αναθέσει τα δεδομένα στους διάφορους co-processors και να

διαχειριστεί τα αποτελέσματα .Έτσι με αυτόν τον τρόπο κάποια P-boxes και S-boxes που είναι κοινά σε κάποιους αλγορίθμους μπορούν να γίνουν γρήγορα σε υλικό αλλά ταυτόχρονα να μπορεί να αλλάξει η λογική του αλγορίθμου .

3.6 Συναφείς εργασίες

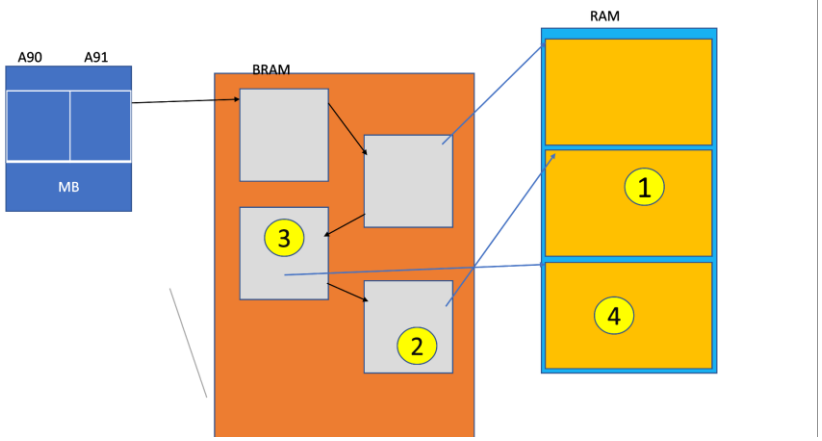
Η δικιά μας εργασία επικεντρώνεται στην προσθήκη ενός soft-core και την ανάπτυξη κώδικα για το sheduling μεταξύ των 2 επεξεργαστών .Μερικές σχετικές εργασίες πάνω σε schedulers και σε αρχιτεκτονική zynq έχουν σκοπό να βελτιώσουν την ασφάλεια καθώς και την απόδοση.Στην εργασία [24] οι συγγράφεις εκμεταλλευόμενοι την συνοχή κρυφής μνήμης (cache-coherent) που μειώνει τον χρόνο επικοινωνίας ανάμεσα στο PS κομμάτι του board και στο PL δοκιμάζουν να επιταχύνουν σε hardware κάποιες εργασίες που γίνονται συχνά και είναι γρήγορες και δεν χρειάζονται πολύ επικοινωνία μεταξύ PS και PL .Μια τέτοιου είδους εργασία είναι και ο scheduler που εκτελείται περιοδικά και είναι μικρή διεργασία σε διάρκεια (fine-grained) και με αυτό τον τρόπο προσπαθούν να επιταχύνουν το schedulng. Επίσης παρόμοια εργασία είναι η [25] οπού για λογούς ασφαλείας μεταφέρουν το schedulng από software σε hardware ώστε να αποκρύψουν ορισμένες λειτουργίες και να κάνουν δυσκολότερο το έργω των επιτιθέμενων .Όσον αφορά την ανταλλαγή κλειδιών στην εργασία [26] ανταλλάσσουν κλειδιά με την μέθοδο Diffie–Hellman αλλά πάνω από ένα μη ασφαλές δίκτυο.

4 Περιγραφή της εργασίας που υλοποιήθηκε

Η βασική λειτουργία του συστήματος είναι η εξής. Υπάρχουν τρεις επεξεργαστές ο Α90 αναθέτει εργασίες, ο Α91 εκτελεί εργασίες και ο ΜΒ που επιλέγει ποια εργασία θα εκτελεστεί δηλαδή κάνει scheduling .Ο πρώτος επεξεργαστής Α90 ξεκινά να βάζει εργασίες στην μνήμη RAM (κοινή μεταξύ Α90 και Α91). Μια εργασία αποτελείται από τρεις πίνακες ,οι πρώτοι δυο πίνακες γεμίζουν με τυχαίους αριθμούς και στον τρίτο πίνακα θα μπουν τα αποτελέσματα αργότερα όταν επεξεργαστούν. Αφού έχει ολοκληρωθεί η αρχικοποίηση μιας εργασίας στην RAM ο Α90 επίσης θα δημιουργήσει και θα προσθέσει έναν αντίστοιχο κόμβο σε μια συνδεδεμένη λίστα η οποία βρίσκεται σε μια άλλη μνήμη την BRAM (κοινή μεταξύ Α90 ,Α91 και ΜΒ). Ο κάθε κόμβος περιέχει πληροφορίες σχετικά με την προτεραιότητα και άλλα δεδομένα για αυτή την εργασία καθώς και έναν δείκτη που δείχνει την θέση των δεδομένων στην RAM . Η συνδεδεμένη λίστα διατρέχεται από τον επεξεργαστή (ΜΒ) που θα βάζει έναν δείκτη στον κόμβο με την μεγαλύτερη προτεραιότητα που επέλεξε. Ο δείκτης που δείχνει την εργασία που επιλέχθηκε βρίσκεται στην κοινή μνήμη BRAM, σε έναν χώρο ειδικά δεσμευμένο για μεταβλητές που χρειάζονται όλοι οι επεξεργαστές να έχουν πρόσβαση .Μετά ο Α91, που η δουλειά του είναι να επεξεργάζεται τα δεδομένα κοιτάζει την θέση μνήμης που βρήκε ο ΜΒ και βρίσκει την αντίστοιχη εργασία που πρέπει να εκτελέσει θα την επεξεργάζεται και θα βάζει το αποτέλεσμα στον τρίτο πίνακα. Για να υλοποιήσουμε το σύστημα ορίσαμε κάποιες καταστάσεις για να γνωρίζουμε σε ποιο στάδιο βρίσκεται κάθε εργασία, στην RAM ορίσαμε 3 καταστάσεις όταν ένα κομμάτι με 3 πίνακες είναι ελεύθερο το συμβολίζουμε με 0 όταν είναι δεσμευμένο και περιμένει να επεξεργαστεί με τιμή 1 και όταν τα αποτελέσματα είναι επεξεργασμένα τότε τα συμβολίζουμε με 2, αντίστοιχα οι κομβόι της λίστας έχουν 2 καταστάσεις, 0 όταν δεν έχουν ακόμα ξεκινήσει να επεξεργάζονται και 1 όταν έχουν ξεκινήσει να επεξεργάζονται. Παρακάτω στο Κεφάλαιο 4.2 θα δούμε που μας χρειάζεται το να κρατάμε αυτή την πληροφορία.

4.1 Παράδειγμα λειτουργίας

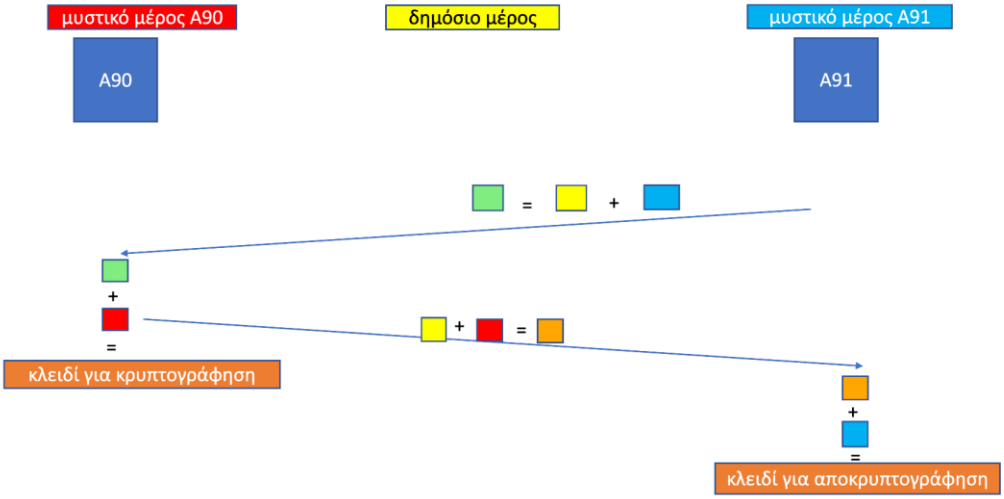
Όπως φαίνεται στο παράδειγμα στην Εικόνα 4.1 αρχικά ο Α90 βρίσκει ένα ελεύθερο κομμάτι στην RAM στο βήμα 1(στον κίτρινο κύκλο) και το αρχικοποιεί κάνοντας την κατάσταση του από 0 που ήταν σε 1 για να δείξει ότι χρησιμοποιείται. Στο βήμα 2 ο Α90 προσθέτει έναν αντίστοιχο κόμβο στην λίστα ,η κατάσταση του κόμβου είναι 0 καθώς δεν έχει αρχίσει η επεξεργασία του ακόμα. Στο βήμα 3 ο scheduler μετά που διέτρεξε όλη την λίστα επέλεξε μια εργασία προς εκτέλεση. Στο βήμα 4 ο Α91 επιλέγει να εκτελέσει την εργασία που δείχνει ο scheduler αλλάζοντας την κατάσταση του κόμβου από 0 σε 1 και έτσι ο scheduler καταλαβαίνει πως έχει αρχίσει η επεξεργασία της εργασίας και θα επιλέξει άλλον κόμβο όσο ο Α91 επεξεργάζεται .Μόλις ο Α91 τελειώσει με την επεξεργασία τότε κάνει την κατάσταση στην RAM ίση με 2 .



Εικόνα 4.1 :Παράδειγμα υλοποίησης

4.2 Ανταλλαγή συμμετρικού κλειδιού ανάμεσα μεταξύ A90 & A91

Για ασφαλή ανάθεση εργασιών ο A90 και A91 ανταλλάσσουν ένα συμμετρικό κλειδί ώστε ο A90 να κρυπτογραφεί τις εργασίες που αναθέτει .Ο αλγόριθμος για την ανταλλαγή συμμετρικού κλειδιού είναι ο Diffie–Hellman όπως τον περιγράψαμε στο Κεφάλαιο 2.7. Στην Εικόνα 4.2 βλέπουμε αυτήν την διαδικασία, ο A91 αρχικά αναμιγνύει το μυστικό μέρος τού κλειδιού του με το δημόσιο μέρος τού κλειδιού και το στέλνει στον A90, μετά ο A90 θα το κρατήσει κάπου ώστε κάθε φορά που θα αναθέτει μια εργασία θα το αναμιγνύει με το μυστικό μέρος του κλειδιού και θα υπολογίσει το συμμετρικό κλειδί που θα κρυπτογραφήσει την εργασία. Για να καταλήξει και ο A91 στο ίδιο κλειδί ο A90 θα αποθηκεύσει το μυστικό του κλειδί που χρησιμοποίησε αναμιγμένο με το δημόσιο μέρος και το τοποθετεί δίπλα στην εργασία που κρυπτογράφησε, όταν ο A91 θα επεξεργαστεί την εργασία θα το αναμίξει με το μυστικό του κλειδί και θα καταλήξει στο ίδιο κλειδί που είχε κρυπτογραφηθεί η εργασία .



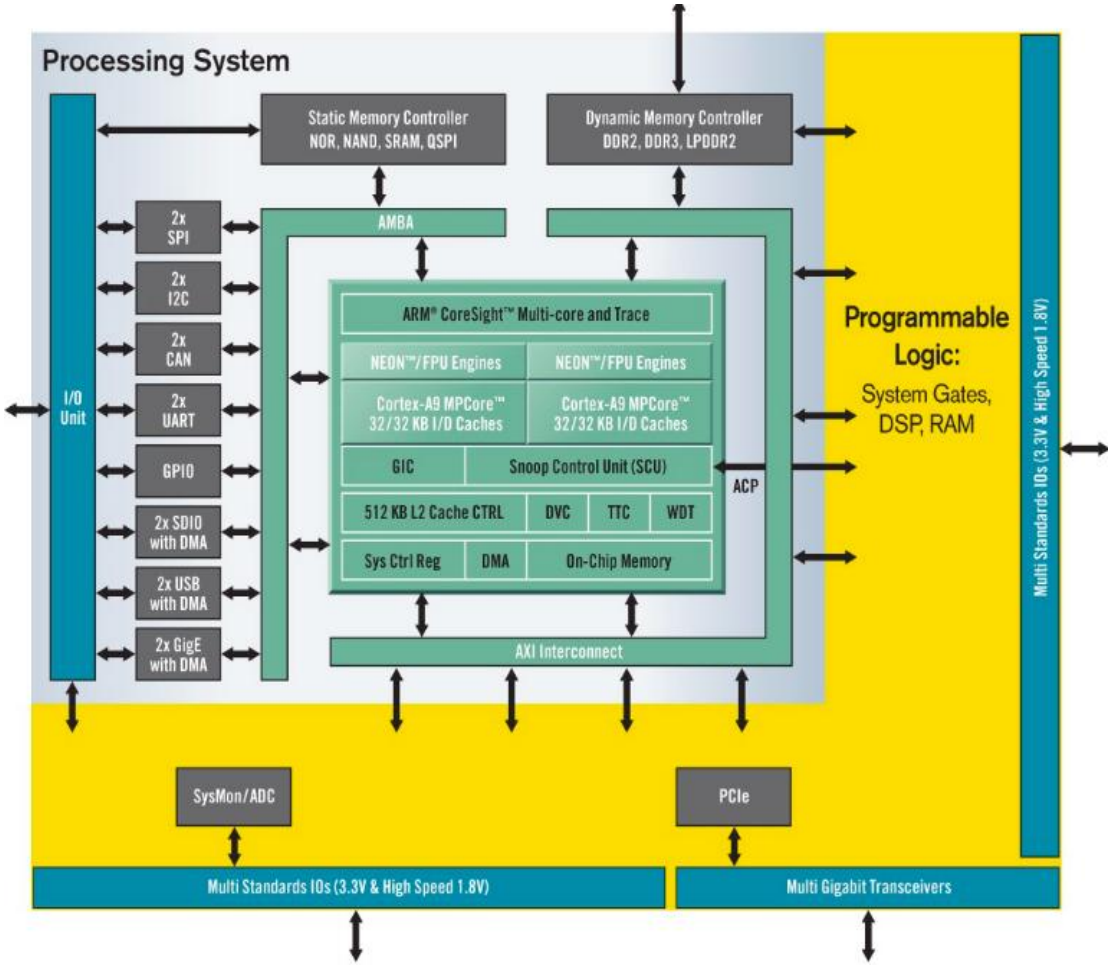
Εικόνα 4.2 :Παράδειγμα ανταλλαγής κλειδιών

5 Περιγραφή υλικού & σχεδιασμός συστήματος

Σε αυτό το κεφάλαιο γίνεται η περιγραφή, η σχεδίαση και η υλοποίηση της εργασίας για το σύστημα που επιλέξαμε καθώς και παραδείγματα λειτουργίας.

5.1 Περιγραφή υλικού που χρησιμοποιήθηκε

Η πλατφόρμα που χρησιμοποιήθηκε για την υλοποίηση είναι το zedboard το οποίο περιλαμβάνει το Zynq®-7000 SoC . Το τελευταίο βασίζεται στην αρχιτεκτονική Xilinx® SoC που ενσωματώνει dual core ARM® CortexTM-A9 processing system (PS) μαζί με programmable logic (PL) οπού στο PL έχουμε προσθέσει έναν microblaze για scheduling και μια μνήμη BRAM για την συνδεδεμένη λίστα και στο (PS) κάνουμε χρήση των δυο επεξεργαστών A9, ο πρώτος επεξεργαστής θα αρχικοποιεί και ο δεύτερος θα εκτελεί εργασίες , στην on-chip μνήμη RAM θα μπαίνουν οι εργασίες .



Εικόνα 5.1: Αρχιτεκτονική zynq

5.1.2 Αρχικοποίηση εκτελέσιμου κώδικα και δεδομένων στις μνήμες του ZYNQ

Η απόφαση για το πού θα τοποθετηθεί ο κώδικας και πού τα δεδομένα στις μνήμες έγινε με βάση τις ανάγκες της εφαρμογής που υλοποιήσαμε έτσι ώστε να δώσουμε τις κατάλληλες ανάλογα με το μέγεθος και την ταχύτητα τους .Οι μνήμες που έχει πρόσβαση κάθε επεξεργαστής φαίνονται στο linkerscript πιο κάτω .Ο A90 και ο A91 έχουν αρχικοποιημένο όλο τον κώδικα τους στην ddr_0 την οποία μοιράσαμε ώστε να μην έχουν πρόβλημα οι επεξεργαστές με τον κώδικα, το heap του A90 όμως όπως θα δούμε το χρειαζόμαστε στην BRAM για να δημιουργήσουμε δυναμικά την συνδεδεμένη λίστα με malloc.

Linkerscript για τον A90 :

```
MEMORY
{
   axi_bram_ctrl_0_Mem0 : ORIGIN = 0x40000000, LENGTH = 0x1FF4
   ps7_ddr_0 : ORIGIN = 0x100000, LENGTH = 0xFF80000
   ps7_qspi_linear_0 : ORIGIN = 0xFC000000, LENGTH = 0x1000000
   ps7_ram_0 : ORIGIN = 0x0, LENGTH = 0x30000
   ps7_ram_1 : ORIGIN = 0xFFFF0000, LENGTH = 0xFE00
}
```

Linkerscript για τον A91:

```
MEMORY
{
   axi_bram_ctrl_0_Mem0 : ORIGIN = 0x40000000, LENGTH = 0x2000
   ps7_ddr_0 : ORIGIN = 0x10080000, LENGTH = 0xFF80000
   ps7_qspi_linear_0 : ORIGIN = 0xFC000000, LENGTH = 0x1000000
   ps7_ram_0 : ORIGIN = 0x0, LENGTH = 0x30000
   ps7_ram_1 : ORIGIN = 0xFFFF0000, LENGTH = 0xFE00
}
```

Όπως βλέπουμε από τα παραπάνω Linkerscript και οι δύο επεξεργαστές βλέπουν την ίδια on-chip μνήμη Ram 0-1 όπου θα μπουν τα δεδομένα για επεξεργασία και έχουν και πρόσβαση στην BRAM στο FPGA.

Αρχικοποίηση για microblaze:

```
MEMORY
{
microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_microblaze_0_local_
memory_dlmb_bram_if_cntlr_Mem : ORIGIN = 0x50, LENGTH = 0x7FB0
axi_bram_ctrl_0_Mem0 : ORIGIN = 0x40000000, LENGTH = 0x2000
}
```

Παραπάνω βλέπουμε τις θέσεις μνήμης που βλέπει ο microblaze. Ο κώδικας βρίσκεται εξ ολοκλήρου στην dlmb μνήμη. Επίσης βλέπει την BRAM καθώς είναι η μνήμη στην οποία θα δημιουργηθεί η συνδεδεμένη λίστα με τις εργασίες που είναι υπεύθυνος για το scheduling και αλλά χρήσιμα δεδομένα που χρειάζεται.

5.1.3 Σχεδιασμός δομών δεδομένων

Για να ολοκληρώσουμε την εργασία χρειάστηκε να αναπτύξουμε κάποιες δομές δεδομένων ώστε να μπορούμε να χειριστούμε κατάλληλα τα δεδομένα που ορίσαμε σε κάθε μνήμη , πιο συγκεκριμένα οι δομές που αναπτύξαμε είναι ,η δομή signal που χρειάζεται στο να κρατάει κάποια δεδομένα που βλέπουν όλοι οι επεξεργαστές , χρησιμοποιήθηκε μια φορά και τοποθετήθηκε στην τέλος σε ένα μικρό κομμάτι της BRAM ώστε να έχουν πρόσβαση όλοι και χρησιμοποιούμε την υπόλοιπη μνήμη BRAM για την συνδεδεμένη λίστα. Τα δεδομένα που κρατάει η δομή είναι το πεδίο *cur το οποίο είναι δείκτης στον κόμβο της λίστας που έχει επιλέξει ο scheduler , το πεδίο head το οποίο είναι δείκτης που δείχνει πάντα στην αρχή της λίστας με τις εργασίες καθώς και ένα ακέραιο pubMixA91 που θα μας χρειαστεί αργότερα στην κρυπτογραφία για την ανταλλαγή κλειδιού .

```
struct signal {
struct Job *cur;
struct Job *head;
int pubMixA91;
};
```

Η δομή job χρησιμοποιείται για να δημιουργηθούν οι κόμβοι της λίστας στην BRAM . Η δομή έχει έναν ακέραιο τον done που κρατάει την πληροφορία για το αν έχει αρχίσει να επεξεργάζεται από τον A91 η εργασία που αντιπροσωπεύει (0= δεν έχει αρχίσει να τα επεξεργάζεται,1 = έχει αρχίσει να τα επεξεργάζεται) , ο ακέραιος αυτός δεν μας λέει αν έχει τελειώσει η επεξεργασία . Η δομή έχει ακόμα , έναν δείκτη *DATApos με την θέση των δεδομένων στην RAM, έναν ακέραιο exe που μας λέει την σειρά που εκτελέστηκε από τον A91 για λογούς αποσφαλμάτωσης και τον συμπληρώνει ο A91 και έναν ακόμα ακέραιο priority για την προτεραιότητα που βοηθάει τον sheduler να επιλέξει την εργασία που θα εκτελεστεί καθώς και έναν δείκτη στην επόμενη δομή της λίστας.

```
struct Job {
int done; //0 1
int exe;
struct blck *DATApos;
int priority;
struct Job *next;
};
```

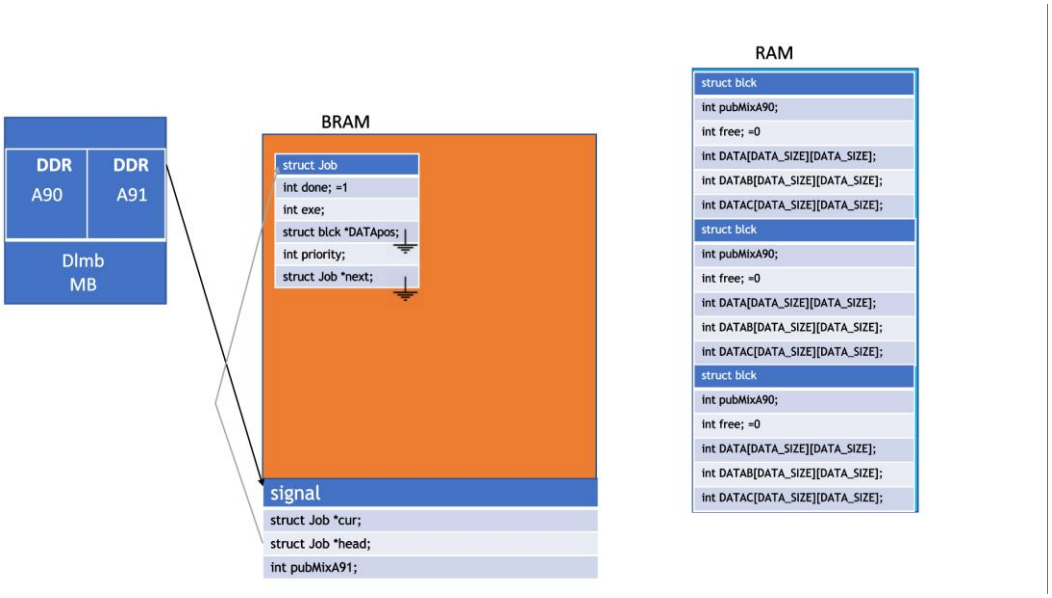
Η δουλειά της δομής blck διαιρεί την RAM σε ίσα κομμάτια όπου το κάθε κομμάτι είναι μια εργασία .Το κάθε κομμάτι περιλαμβάνει έναν ακέραιο τον οποίο θα χρησιμοποιήσουμε στην κρυπτογραφία, ένα flag που μας λέει την κατάσταση του κομματιού της μνήμης (0= ελεύθερο για χρήση ,1 = δεσμευμένο - δεν έχει επεξεργαστεί, 2 = επεξεργασμένο) και τρεις πίνακες με δεδομένα που θα επεξεργαστούν, εκ των οποίων οι πρώτοι δυο πίνακες είναι οι πίνακες που θα πολλαπλασιαστούν και ο τρίτος είναι για το αποτέλεσμα.

```
struct blck {
int pubMixA90;
```

```
int free;//0 1 2
int DATA[DATA_SIZE][DATA_SIZE];
int DATAB[DATA_SIZE][DATA_SIZE];
int DATAC[DATA_SIZE][DATA_SIZE];
};
```

5.1.4 Συνολική εικόνα του συστήματος

Στην Εικόνα 5.2 υπάρχει το σύστημα όπως το ορίσαμε στα κεφάλαια 5.1.1-3 . Όπως αναφέραμε πιο πάνω η δομή signal τοποθετήθηκε στο κάτω μέρος της BRAM ώστε να μπορούν να έχουν όλοι πρόσβαση στα δεδομένα που κρατάει. Το πορτοκαλί μέρος της BRAM είναι που θα δημιουργηθεί η λίστα με της εργασίες οπού ο A90 δεσμεύει δυναμικά χώρο με την δομή Job. Η RAM έχει χωριστεί σε κομμάτια με την δομή blck .



Εικόνα 5.2: Συνολική εικόνα του συστήματος

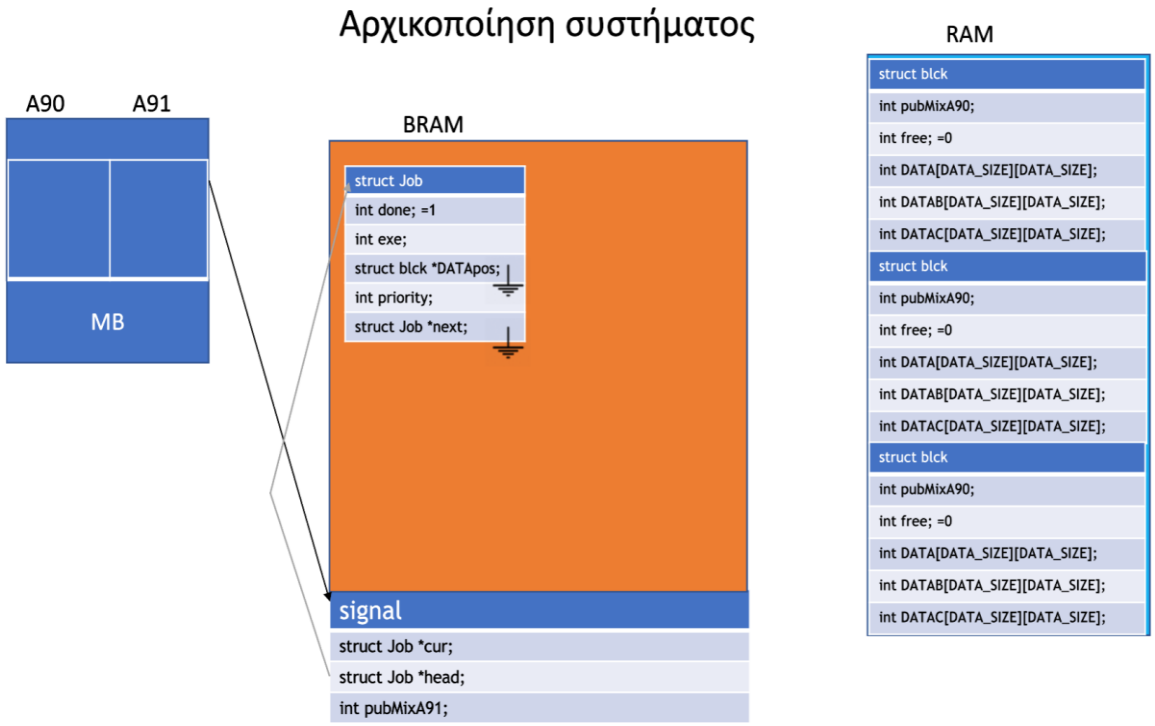
5.2 Περιγραφή λειτουργίας συστήματος

Όπως περιγράψαμε και σε προηγούμενα κεφάλαια η λειτουργία του συστήματος χωρίζεται σε 3 βασικές υπο-λειτουργίες ,προσθήκη εργασιών από τον A90 επιλογή από τον MB και εκτέλεση από τον A91 ,πριν όμως μπορέσουμε να προσθέσουμε εργασίες και

χρησιμοποιήσουμε το σύστημα πρέπει να γίνει κάποια αρχικοποίηση στην κοινές μεταβλητές και μνήμες όπως θα δούμε παρακάτω .

5.2.1 Αρχικοποίηση συστήματος

Το πρώτο βήμα για να λειτουργήσει το σύστημα είναι ο A90 να αρχικοποίηση τα flags και κάποιες κοινές μεταβλητές που βασίζονται οι επεξεργαστές για την λειτουργία τους . Ο A90 φτιάχνει για πρακτικούς λογούς τον πρώτο κόμβο (βλ. Παράρτημα Α 187-193) που δεν αντιστοιχεί σε κάποια εργασία και δεν διαγράφεται ποτέ και βάζει done =1 ώστε να μην τον επιλέξει ο scheduler .Το πεδίο next αρχικοποιείται σε NULL για να δηλώσει το τέλος της λίστας όπου θα μπει ο δείκτης στην πρώτη δουλειά αργότερα .Επίσης αρχικοποιεί το πεδίο head στην δομή signal ώστε να δείχνει αυτόν τον κόμβο. Τέλος με μια επανάληψη αρχικοποιούνται όλα τα πεδία free στην RAM σε 0 έτσι ώστε να είναι δηλωμένα ελεύθερα για να μπουν τα δεδομένα προς επεξεργασία (βλ. Παράρτημα Α 193-197).

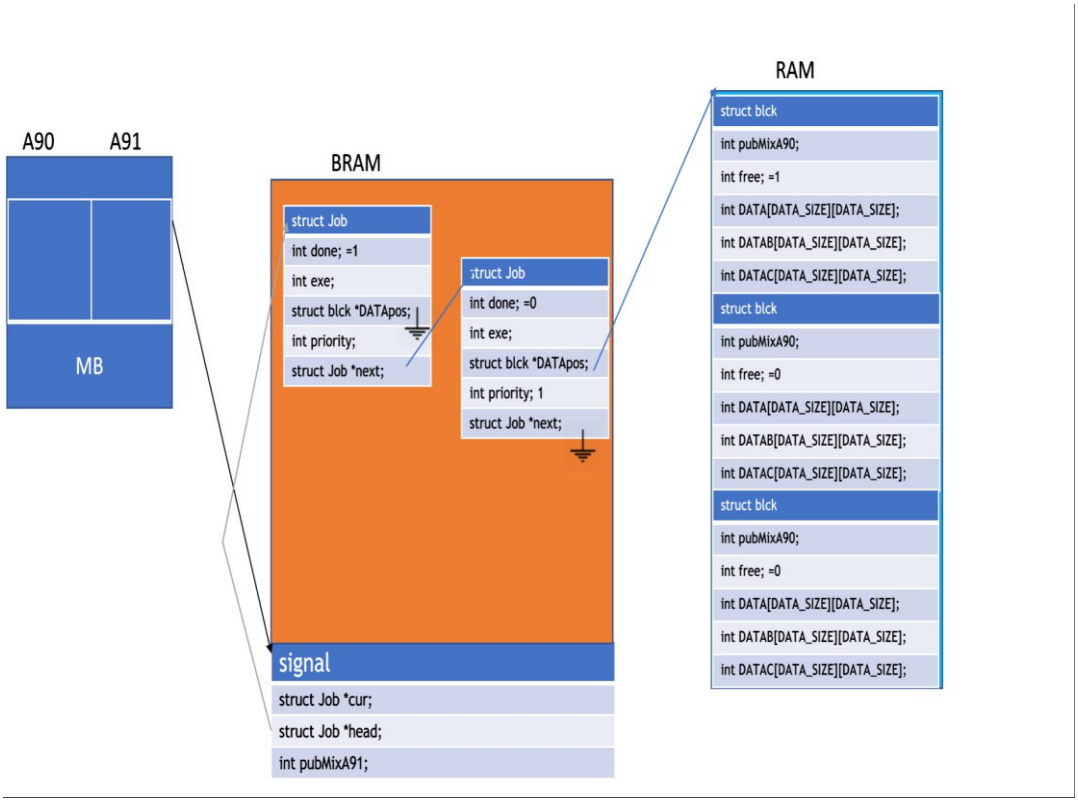


Εικόνα 5.3: Αρχικοποιημένο σύστημα

5.2.2 Εισαγωγή εργασιών

Αφού ο A91 εκτελέσει τις αρχικοποιήσεις που περιγράψαμε στο κεφάλαιο 5.2.1 το σύστημα είναι έτοιμο να λειτουργήσει και ο A90 αρχίζει να προσθέτει εργασίες. Αρχικά ο A90 θα ψάξει όλα τα πεδία free στη RAM μέχρι να βρει κάποιο με τιμή 0 αν δεν βρει επιστρέφει σφάλμα και δεν προσθέτει την εργασία . Μόλις βρει κάποιο ελεύθερο κομμάτι θα αρχικοποιήσει τους 2 πρώτους πίνακες και θα κάνει το πεδίο free=1 για να δηλώσει ότι είναι

πιασμένο . Έπειτα εφόσον έχει δημιουργηθεί μια εργασία στην RAM με επιτυχία ο A90 θα δημιουργήσει και έναν αντίστοιχο κόμβο στην BRAM με τα απαραίτητα δεδομένα για την συγκεκριμένη εργασία όπως περιγράψαμε πιο πάνω, δηλαδή θα βρει δυναμικά ένα κομμάτι μνήμης με malloc ,θα βάλει προτεραιότητα τον δείκτη στα δεδομένα που αντιστοιχούν στην RAM και θα κάνει τον τελευταίο κόμβο της λίστας να δείχνει στον καινούργιο κόμβο που έφτιαξε όπως στην Εικόνα 5.4 (βλ. Παράρτημα A 206-224).

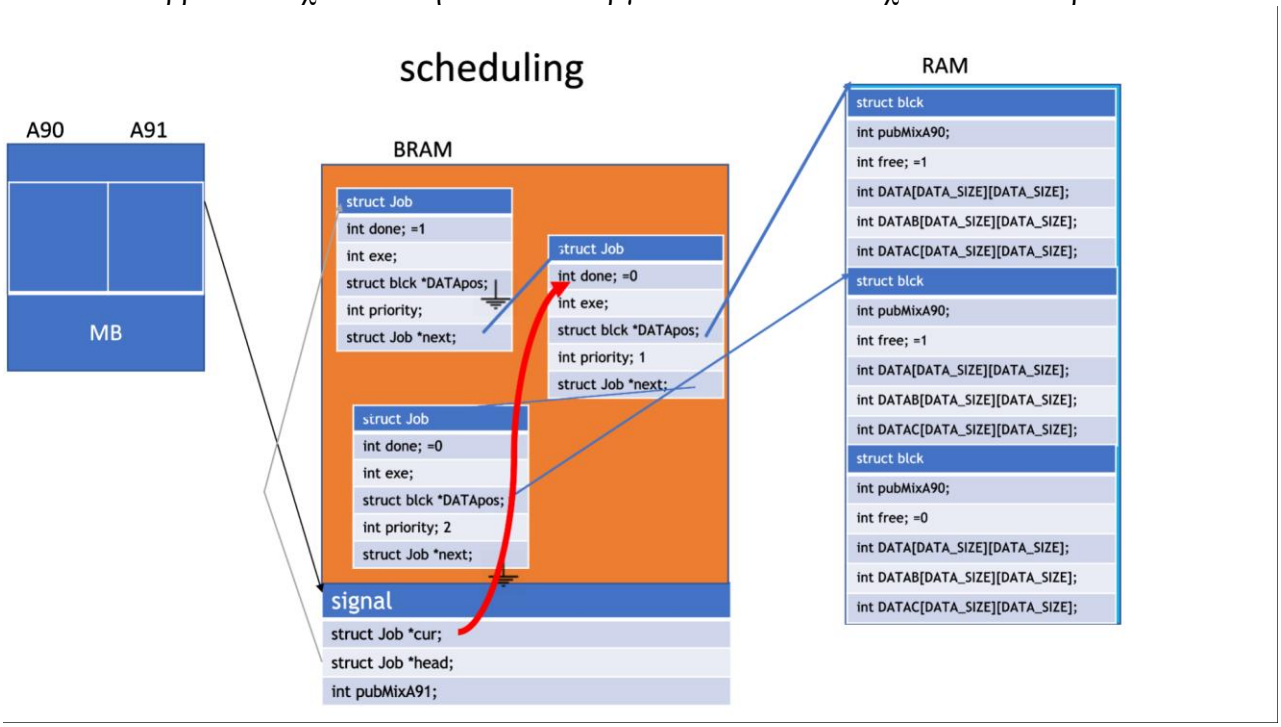


Εικόνα 5.4: Εισαγωγή εργασιών

5.2.3 Scheduling

Το scheduling γίνεται επαναληπτικά από τον microblaze από την στιγμή που ο A90 έχει αρχικοποιήσει το σύστημα στο κεφάλαιο 5.2.1 .Ο microblaze διατρέχει συνεχώς την λίστα και στο τέλος κάθε ενός περάσματος τις λίστας βάζει τον δείκτη *cur να δείχνει στην εργασία που επέλεξε (βλ. Παράρτημα A 510 -522) .Η επιλογή γίνεται με το να κοιτάζει να βρει μια εργασία για την οποία ισχύουν δυο πράγματα ,πρώτον να έχει την μεγαλύτερη προτεραιότητα από τις υπόλοιπες και δεύτερον αν το done της εργασίας αυτής είναι 0 αλλιώς

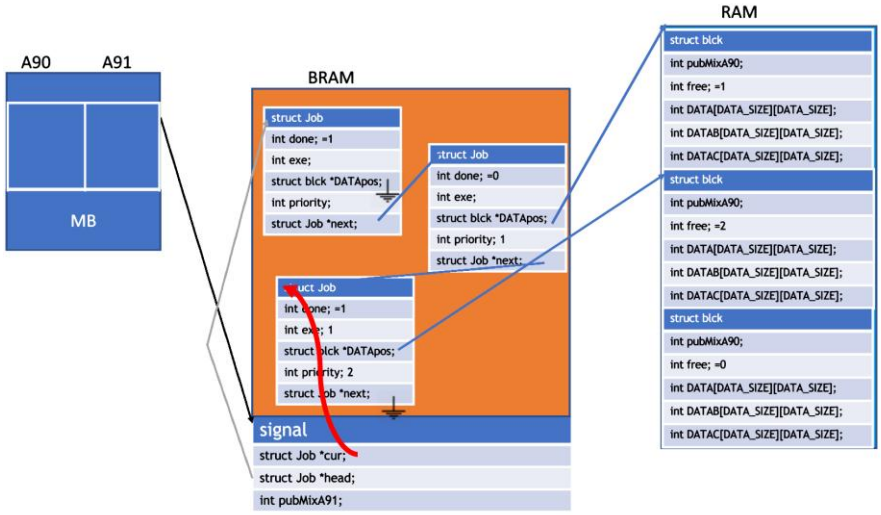
αν δεν βρει δείχνει στην ίδια εργασία που έδειχνε και πριν .



Εικόνα 5.5: Επιλογή εργασίας για εκτέλεση

5.2.4 Επεξεργασία & συλλογή αποτελεσμάτων

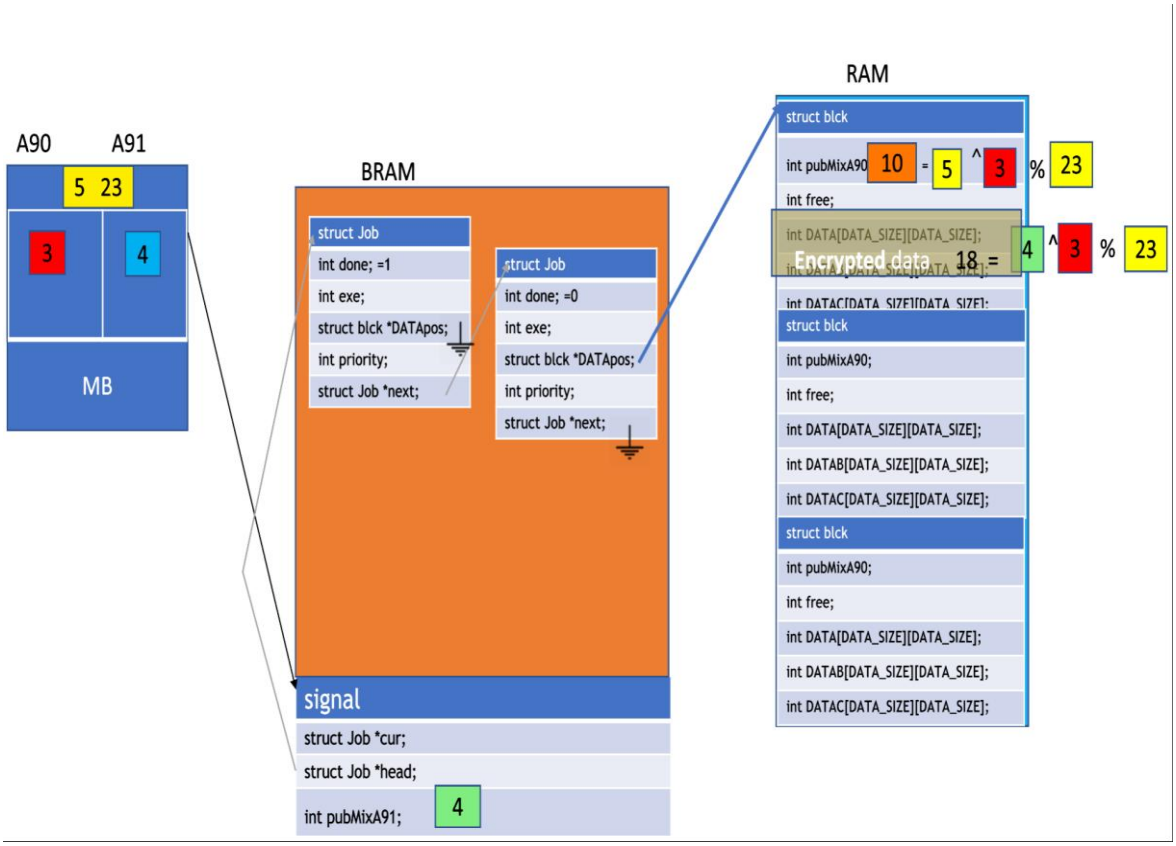
Ο A91 πάει μέσω του δείκτη signal->cur και αφού ελέγξει ότι στην εργασία που δείχνει ο δείκτης *cur ισχύει done ίσο με 0 και free ίσο με 1 δηλαδή η εργασία που δείχνει είναι καινούργια τότε την επεξεργάζεται. Η πρώτη δουλειά του A91 είναι να κάνει το done ίσο με 1 ώστε ο scheduler να καταλάβει πως ο A91 έχει πάρει και επεξεργάζεται την εργασία, με αυτόν το τρόπο ο sheduler θα ψάχνει για την επόμενη εργασία που θα εκτελεστεί όσο ο A91 θα επεξεργάζεται (βλ. Παράρτημα Α 417-472). Μόλις τελειώσει με την επεξεργασία ο A91 κάνει το free = 2 ώστε ο A90 να ξέρει ότι τελείωσε ώστε να μαζέψει τα αποτελέσματα και κοιτάζει για την επόμενη εργασία που του έστειλε ο scheduler (βλ. Παράρτημα Α 228-280).



Εικόνα 5.6: Εκτέλεση εργασίας

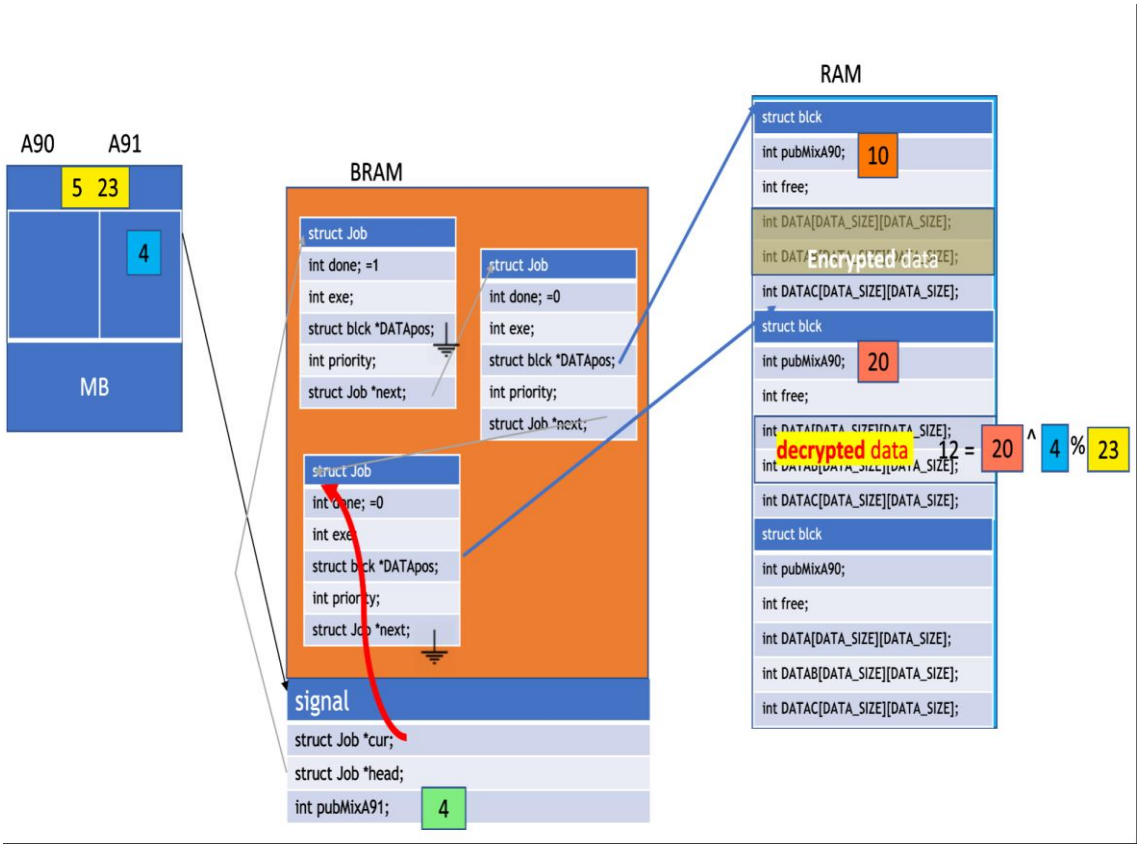
5.3 Υλοποίηση ανταλλαγής κλειδιού Diffie–Hellman και κρυπτογραφία AES-ECB

Παρακάτω στην Εικόνα 5.7 βλέπουμε πως υλοποιήσαμε την κρυπτογραφία και ανταλλαγή κλειδιού μεταξύ των δύο επεξεργαστών. Αρχικά ο A91 αναμιγνύει το ιδιωτικό μέρος του κλειδιού με το δημόσιο μέρος και το αποτέλεσμα το τοποθετεί στην δομή signal ,με αυτό τον τρόπο ο A90 μπορεί να το χρησιμοποιήσει. Το επόμενο βήμα είναι ο A90 να δημιουργήσει ένα τυχαίο αριθμό που θα τον χρησιμοποιήσει ως μυστικό μέρος του κλειδιού μόνο για μια εργασία. Μετά θα αναμίξει το ιδιωτικό του κλειδί με αυτό που άφησε ο A91 στην δομή signal και καταλήγει στο συμμετρικό κλειδί που θα χρησιμοποιήσει για να κρυπτογραφήσει μια εργασία. Τέλος ο A90 αναμιγνύει το μυστικό μέρος του κλειδιού του με το δημόσιο μέρος του κλειδιού και αφήνει το αποτέλεσμα στη δομή blck που βρίσκεται στο πεδίο pubMixA90 κάθε εργασίας, έτσι ώστε όταν έρθει η ώρα να εκτελεστεί η εργασία ο A91 θα το αναμείξει με το μυστικό μέρος του κλειδιού του και έτσι θα καταλήξει στο συμμετρικό κλειδί που κρυπτογραφήθηκε η εργασία .



Εικόνα 5.7: Δημιουργία κλειδιού & κρυπτογράφηση δεδομένων

Όταν ο A91 θέλει να επεξεργαστεί τα δεδομένα χρησιμοποιεί την πληροφορία που άφησε ο A90 στην δομή blck και φτιάχνει το συμμετρικό κλειδί με τον τρόπο που φαίνεται στην Εικόνα 5.8 αναμιγνύοντας τον ακέραιο που άφησε ο A90 στο πεδίο pubMixA90 με το ιδιωτικό του κλειδί και με αυτό τον τρόπο καταλήγουν στο ίδιο κλειδί που είχε χρησιμοποιήσει και ο A90 για την κρυπτογραφία.



Εικόνα 5.8: Αποκρυπτογράφηση και επεξεργασία

6 Αποτελέσματα και μετρήσεις

6.1 Μετρήσεις

Για να βεβαιωθούμε ότι το σύστημα λειτουργεί σύμφωνα με τις προδιαγραφές που περιγράψαμε πιο πάνω στο κεφαλαίο 4 το ελέγξαμε δίνοντας του να φέρει εις πέρας έναν αριθμό εργασιών σε πειράματα με διάφορες προτεραιότητες και μετρήθηκε η καθυστέρηση σε κύκλους ρολογιού σε κάποια συγκεκριμένα σημεία του κώδικα για να μας βοηθήσει να κατανοήσουμε και να εξηγήσουμε την συμπεριφορά του συστήματος.

Τα σημεία που μας ενδιαφέρει να μετρήσουμε στον A90 είναι, ο χρόνος που θέλει να προσθέσει μια εργασία και μπορεί να χωριστεί σε 3 στάδια ,το πρώτο είναι να βρει μια άδεια θέση στην RAM για τους 3 πίνακες και είναι μεταβλητός ανάλογα με το ποσό γρήγορα θα βρει μια θέση, το δεύτερο στάδιο είναι να αρχικοποιήσει την θέση εδώ ο χρόνος είναι πάντα σταθερός και εξαρτάται από το μέγεθος των πινάκων και αν χρησιμοποιεί κρυπτογράφηση και το τρίτο στάδιο είναι ο χρόνος να προσθέσει έναν κόμβο στην λίστα που είναι μεταβλητός και εξαρτάται από το μέγεθος της λίστας που πρέπει να διατρέξει .Στον A91 οι χρόνοι είναι σταθεροί εφόσον έχει έναν δείκτη έτοιμο στα δεδομένα που θα επεξεργαστεί και κάνει 2 λειτουργίες αποκρυπτογράφηση και επεξεργασία .Όσον αφορά τον χρόνο του scheduling ο χρόνος που χρειάζεται για να διατρέξει την λίστα ο MB είναι συγκριτικά πολύ μικρότερος σε σχέση με τις υπόλοιπες λειτουργίες των άλλων επεξεργαστών και επιπλέον μόλις ο A91 αρχίζει να επεξεργάζεται μια εργασία ο scheduler τρέχει παράλληλα για να βρει την επόμενη οπότε προλαβαίνει να διατρέξει πολλές φορές την λίστα με τις εργασίες όσο ο A91 επεξεργάζεται οπότε δεν μας επηρεάζει όσον αφορά την επίδοση του συστήματος .

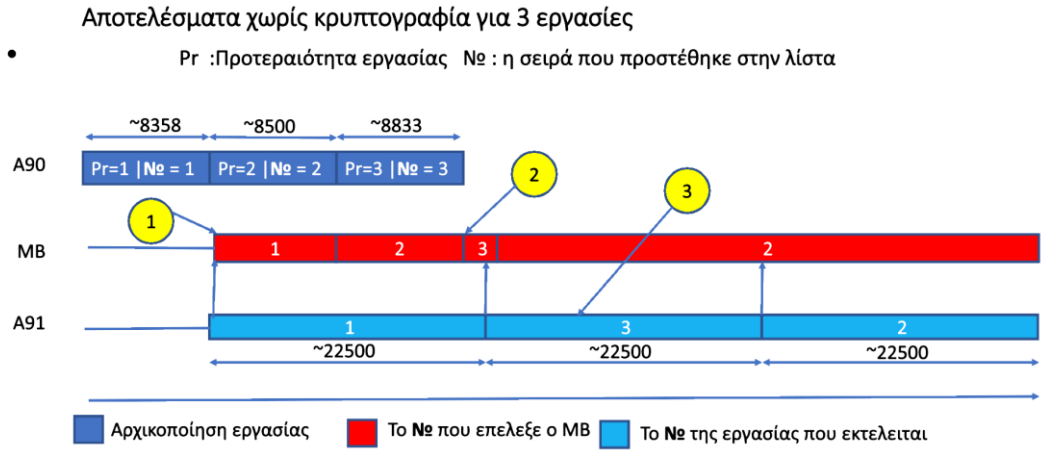
6.2 Αποτελέσματα

6.2.1 Δοκιμή ορθής λειτουργίας του συστήματος χωρίς κρυπτογραφία

Από τις μετρήσεις που έγιναν συμπεράναμε ότι για πειράματα με μέχρι 50 εργασίες για να προστεθεί μια εργασία στην συνδεδεμένη λίστα και να αρχικοποιήσει τα αντίστοιχα δεδομένα στην RAM ο A90 χρειάζεται από ~8500 κύκλους μέχρι ~26000 ανάλογα με το πόσο χρόνο θα χρειαστεί να βρει μια άδεια θέση όπως περιγράψαμε στο κεφάλαιο 6.1 .Ο A91 χρειάζεται ~22000 σταθερά κύκλους για να επεξεργαστεί τα δεδομένα .

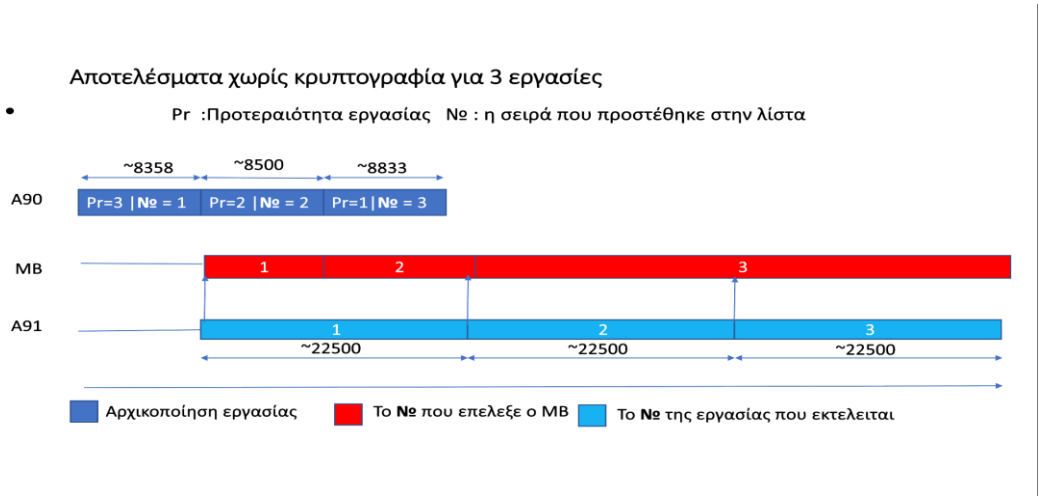
Παρακάτω στην Εικόνα 6.1 βλέπουμε τα αποτελέσματα για 3 εργασίες που τρέξαμε με αύξουσα προτεραιότητα δηλαδή κάθε εργασία που προσθέτουμε έχει μεγαλύτερη προτεραιότητα από την προηγούμενη .Στην αρχή της λειτουργίας του συστήματος παρατηρούμε ότι ο A90 αρχίζει να αρχικοποιεί ενώ ο sheduler παρακολουθεί για το πότε θα ολοκληρωθεί η αρχικοποίηση ώστε να περάσει τον δείκτη στον A91 και να αρχίσει την επεξεργασία .Στο σημείο 1 έχει ολοκληρωθεί η αρχικοποίηση της εργασίας 1 και ο sheduler δίνει κατευθείαν την επεξεργασία στην πρώτη εργασία .Λίγο αργότερα αναμεσά στο σημείο

1 και 2 έχουν προλάβει να αρχικοποιηθούν 2 ακόμα εργασίες η 2 και η 3 όσο ακόμα ο A91 εκτελεί την εργασία 1, οπότε ο scheduler επιλέγει την εργασία 3 στο σημείο 2 την οποία ο A91 εκτελεί μόλις τελειώσει με την εργασία 1 ,μόλις ο A91 αρχίζει να εκτελεί την εργασία 3 ο scheduler αρχίζει να ψάχνει την επόμενη εργασία που θα εκτελεστεί που είναι η 2 επειδή δεν υπάρχουν άλλες .



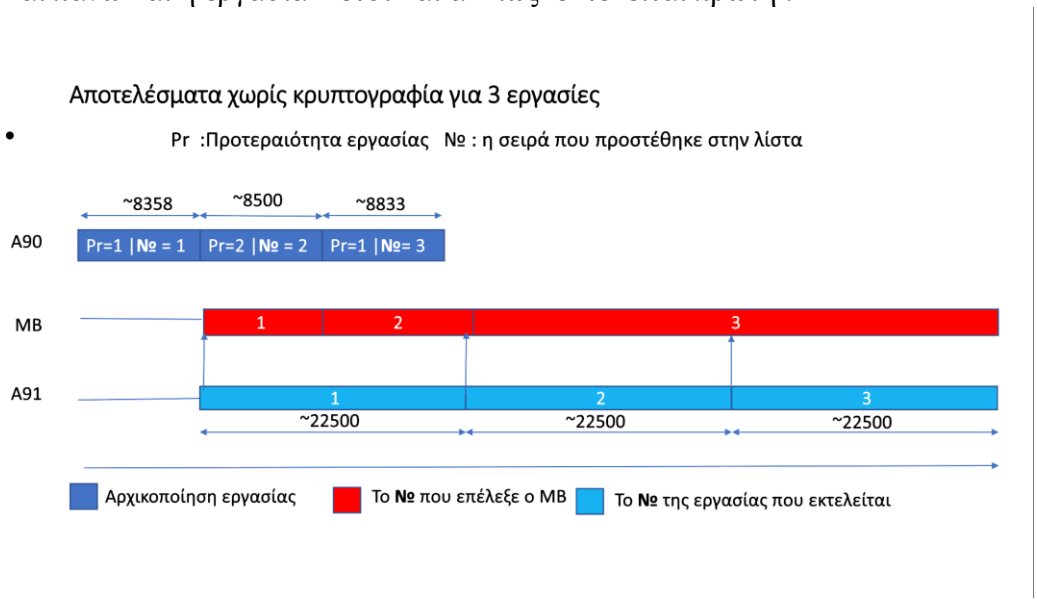
Εικόνα 6.1: Χρονοδιάγραμμα για 3 εργασίες με αύξουσα προτεραιότητα

Στην Εικόνα 6.2 κάνουμε το ίδιο πείραμα αλλά τώρα με φθίνουσες προτεραιότητες δηλαδή κάθε εργασία που προστίθεται έχει μικρότερη προτεραιότητα από την προηγούμενη. Παρατηρούμε ότι έχει παρόμοια συμπεριφορά στην αρχή αλλά με την διαφορά ότι όταν τελειώσει η αρχικοποίηση της εργασίας 3 δεν την επιλέγει ο scheduler γιατί έχει μικρότερη προτεραιότητα από την εργασία 2 οπότε εξακολουθεί να δείχνει την εργασία 2 μέχρι που ο A91 αρχίζει να την εκτελεί και μετά ο scheduler βρίσκει την εργασία 3.



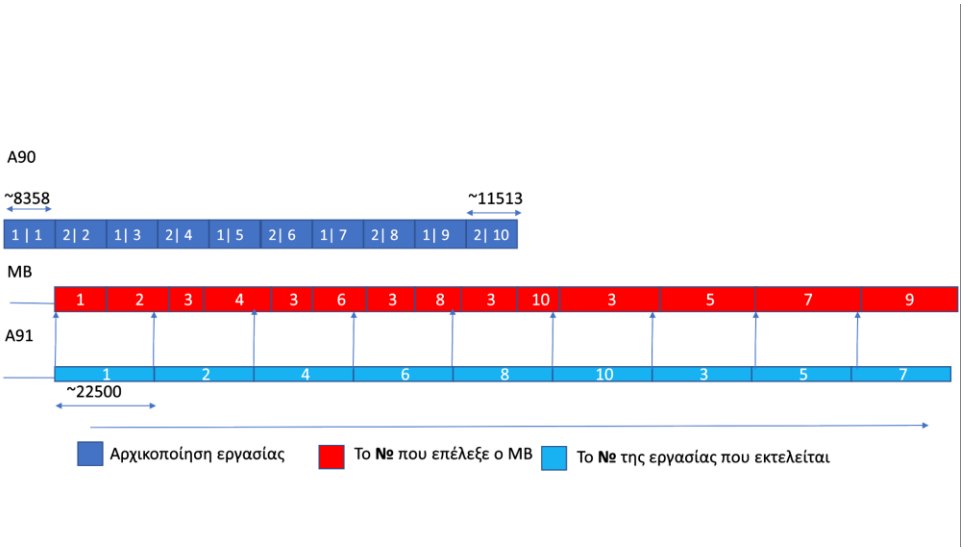
Εικόνα 6.2: Χρονοδιάγραμμα για 3 εργασίες με φθίνουσα προτεραιότητα

Στην Εικόνα 6.3 κάναμε το ίδιο πείραμα αλλά με εκ περιτροπής προτεραιότητες δηλαδή η πρώτη θα έχει 1 η δεύτερη 2 η τρίτη 1 κτλ. παρατηρούμε ότι συμβαίνει κάτι παρόμοιο με την Εικόνα 6.2 καθώς η εργασία 2 έχει μεγαλύτερη προτεραιότητα από την 3 όπως και πάνω και η εργασία 1 έτσι και αλλιώς εκτελείται πρώτη .



Εικόνα 6.3:Χρονοδιάγραμμα για 3 εργασίες με προτεραιότητα εκ περιτροπής

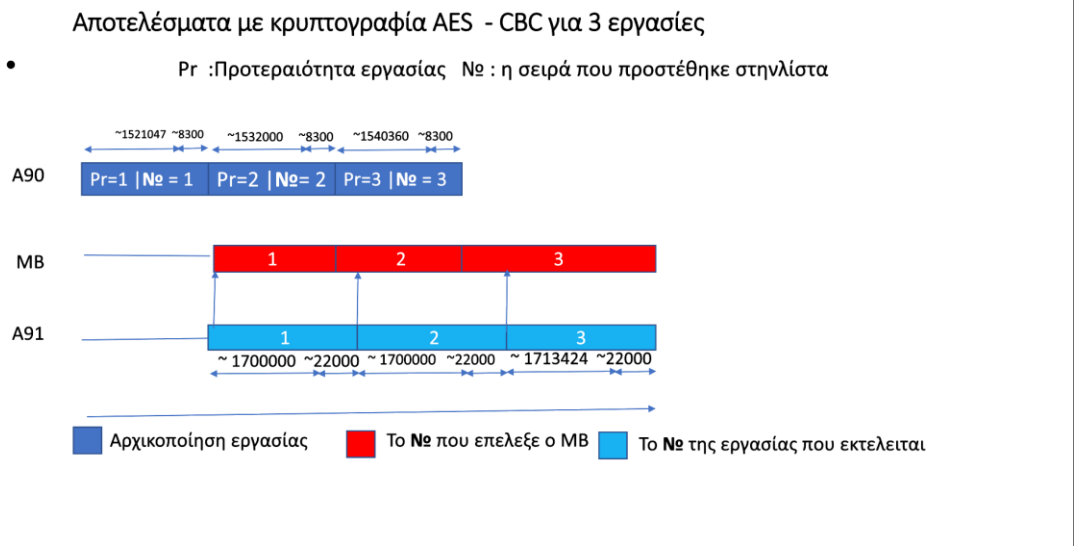
Στην Εικόνα 6.4 δοκιμάσαμε το σύστημα για 10 εργασίες με εκ περιτροπής προτεραιότητες. Παρατηρούμε ότι εδώ και πάλι η εργασία 1 στέλνεται και εκτελείται πρώτη καθώς δεν υπάρχουν άλλες εργασίες .Στο χρονικό όμως διάστημα που επεξεργάζεται η εργασία 1 έχουν προλάβει να αρχικοποιηθούν άλλες δυο εργασίες ,η εργασία 2 με 2 προτεραιότητα και η εργασία 3 με 1 προτεραιότητα οπότε ο sheduler στέλνει την εργασία 2 για εκτέλεση ,μετά στο διάστημα που εκτελείται μια εργασία με προτεραιότητα 2 ο A90 προλαβαίνει να αρχικοποιήσει τουλάχιστον άλλη μια εργασία με προτεραιότητα 2 οπότε όλες η εργασίες με προτεραιότητα 1 μένουν για το τέλος και εκτελούνται αυτές με την 2 προτεραιότητα μέχρι να τελειώσει και η εργασία 10 που είναι η τελευταία που προστίθεται , μετά ακολουθούν οι υπόλοιπες εργασίες που δεν εκτελέστηκαν με προτεραιότητα 1. Παρατηρούμε εδώ επίσης ότι ο sheduler επιλέγει ανά τακτά διαστήματα την εργασία 3 που έχει προτεραιότητα 1 αλλά προλαβαίνουν να αρχικοποιηθούν εργασίες με προτεραιότητα 2.



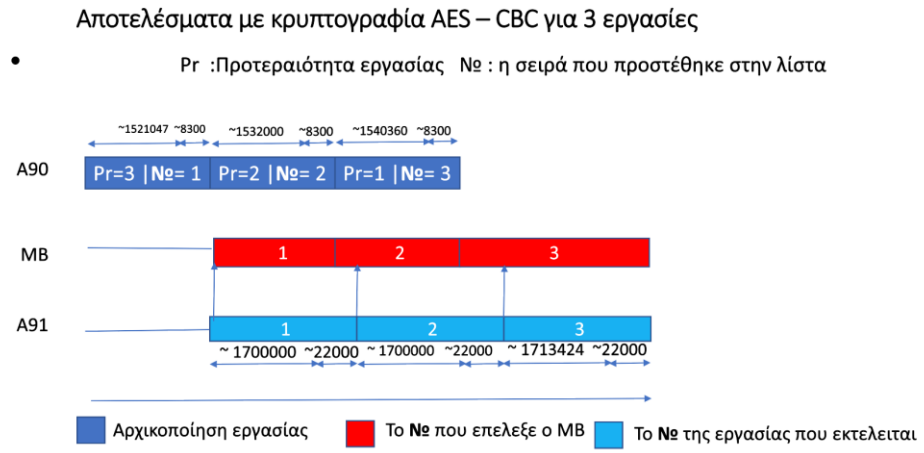
Εικόνα 6.4: Χρονοδιάγραμμα για 10 εργασίες με προτεραιότητα εκ περιτροπής

6.2.2 Δοκιμή ορθής λειτουργίας του συστήματος με κρυπτογραφία

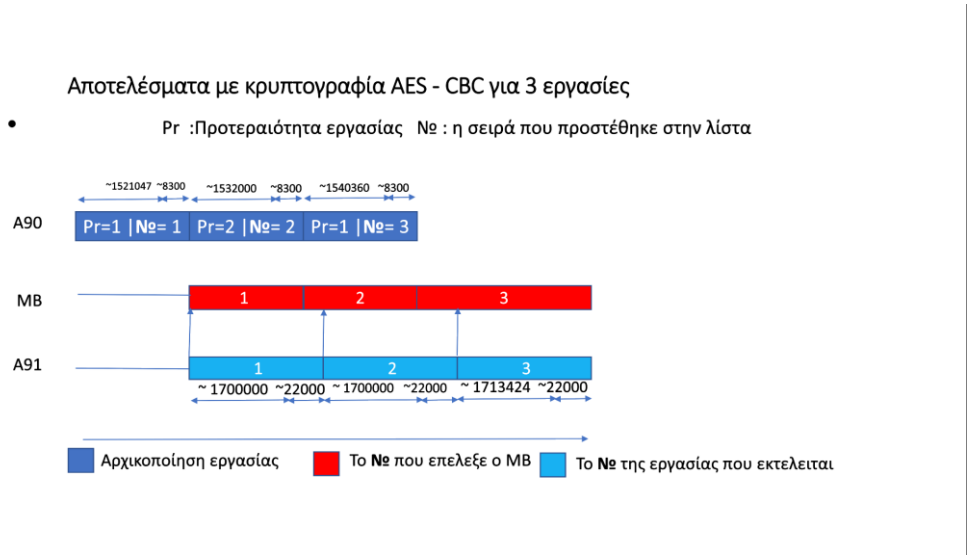
Με την προσθήκη κρυπτογραφίας AES-ECB η δημιουργία κάποιου κόμβου και η αρχικοποιήση των δεδομένων τώρα χρειάζεται μεταξύ 1526400 – 1560390 κύκλους αντίστοιχα. Ο A91 για να αποκρυπτογραφήσει και να επεξεργαστεί μια εργασία παίρνει ~1728617 κύκλους .Τα αντίστοιχα πειράματα με το προηγούμενο Κεφάλαιο 6.2.1 είναι τα παρακάτω .Εδώ παρατηρούμε ότι με την προσθήκη κρυπτογραφίας οι χρόνοι αρχικοποίησης και εκτέλεσης είναι σχεδόν οι ίδιοι οπότε κάθε εργασία που αρχικοποιείται στέλνεται για εκτέλεση κατευθείαν μόλις τελειώσει η προηγουμένη.



Εικόνα 6.5: Χρονοδιάγραμμα για 3 εργασίες με αύξουσα προτεραιότητα και κρυπτογραφία



Εικόνα 6.6: Χρονοδιάγραμμα για 3 εργασίες με φθίνουσα προτεραιότητα και κρυπτογραφία



Εικόνα 6.7: Χρονοδιάγραμμα για 3 εργασίες με προτεραιότητα εκ περιτροπής προτεραιότητα με κρυπτογραφία

7 Μελλοντική Εργασία και συμπεράσματα

7.1 Συμπεράσματα

Από τα παραπάνω μπορούμε να συμπεράνουμε ότι η ασφάλεια είναι ένα σημαντικό κομμάτι ειδικά για εφαρμογές που χειρίζονται προσωπικά και ευαίσθητα δεδομένα που θα μπορούσαν να θέσουν την ιδιωτικότητα ή ακόμα και τις ζωές μας σε κίνδυνο, αλλά η εφαρμογή τέτοιων τεχνικών σε συσκευές επηρεάζει την επίδοση τους, ειδικά σε ενσωματωμένα συστήματα.

Όσον αφορά την εργασία αναπτύξαμε με επιτυχία ένα σύστημα που εκτελεί εργασίες με προτεραιότητα και καταφέραμε να συνδυάσουμε και κρυπτογραφία ώστε να αναπτύξουμε ένα ασφαλές σύστημα ,η τεχνική δουλεύει αρκετά καλά αν το μυστικό μέρος του κλειδιού παραμένει πάντα κρυφό. Σχετικά με την κρυπτογράφηση παρατηρήσαμε πως αυξάνει σημαντικά το overhead όμως κερδίζουμε σε ασφάλεια.

7.2 Μελλοντική Εργασία

Στην παρούσα εργασία υλοποιήσαμε το κομμάτι όπου ο ένας επεξεργαστής αναθέτει στον άλλον εργασίες να εκτελέσει με ασφάλεια αλλά υπάρχουν ακόμα μερικά θέματα που πρέπει να λάβουμε υπόψη στο μέλλον για να κάνουμε πιο ολοκληρωμένη την εργασία , θα μπορούσαμε να υλοποιήσουμε μια μέθοδο οπού ο πρώτος επεξεργαστής θα παίρνει πίσω με ασφάλεια τα αποτελέσματα κρυπτογραφημένα ,επίσης μπορούμε να βρούμε μια λύση για πιο σωστή διαχείριση των κλειδιών ώστε να μην αποθηκεύονται στην κοινή μνήμη οπού όλοι μπορούν να έχουν πρόσβαση . Μπορούμε να κάνουμε αλλαγές και όσον αφορά το κομμάτι της απόδοσης και διαχείρισης των εργασιών που καταφθάνουν . Όσον αφορά την απόδοση, για παράδειγμα, θα μπορούσαμε να αντικαταστήσουμε την χρονοβόρα διαδικασία της κρυπτογράφησης με τεχνικές που αναφέραμε πιο πάνω, στο κεφαλαίο 3 με ελαφριά κρυπτογράφηση από λογισμικό η/και υλικό ώστε να πετύχουμε καλύτερους χρόνους στο κομμάτι αυτό. Επιπλέον αφού κάνουμε το παραπάνω τότε ίσως θα χρειαστεί και το scheduling να γίνει πιο γρήγορο για να προλαβαίνει την όλη διαδικασία. Αυτό μπορεί να γίνει αν υλοποιήσουμε τον scheduler σε μια γλωσσά περιγραφής υλικού, με αποτέλεσμα να πετύχουμε καλύτερους χρόνους, ή να αλλάξουμε την δομή δεδομένων από μια συνδεδεμένη λίστα σε κάποιο δυαδικό δέντρο με βάση τις προτεραιότητες των εργασιών. Όσον αφορά την διαχείριση των εργασιών θα μπορούσαμε να δημιουργήσουμε δυναμικές προτεραιότητες ώστε να μην έχουμε εργασίες που θα περιμένουν για πάντα όταν καταφθάνουν συνεχεία εργασίες με μεγαλύτερη προτεραιότητα. Μια ακόμα ιδέα για να βελτιώσουμε το σύστημα είναι, αν ο scheduler βρει μια εργασία με μεγαλύτερη προτεραιότητα, αντί να περιμένει να τελειώσει η ήδη υπάρχουσα που εκτελείται να παραχωρήσει την επεξεργασία στην άλλη με τη μεγαλύτερη προτεραιότητα και να την εκτελέσει αφήνοντας κάποια πληροφορία για το σημείο που σταμάτησε.

Βιβλιογραφικές αναφορές

[1] G. Kornaros, O. Tomoutzoglou, D. Mbakoyiannis, N. Karadimitriou, M. Coppola, E. Montanari, I. Deligiannis, G. Gherardi, "Towards Holistic Secure Networking in Connected Vehicles through Securing CAN-bus Communication and Firmware-over-the-Air Updating", Journal of Systems Architecture (2020), vol. 109, pp. 101761

[2] G. Kornaros, D. Bakoyiannis, O. Tomoutzoglou, M. Coppola and G. Gherardi, "TrustNet: Ensuring Normal-world and Trusted-world CAN-bus Networking," 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), Beijing, China, 2019, pp. 1-6.

[3] D. Mbakoyiannis, O. Tomoutzoglou, and G. Kornaros, "Secure Over-the-air Firmware Updating for Automotive Electronic Control Units", Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), pp. 174—181, Limassol, Cyprus, 2019

[4] G. Kornaros, I. Christoforakis, O. Tomoutzoglou, D. Bakoyiannis,K. Vazakopoulou, M. Grammatikakis, A. Papagrigoriou, "Hardware Support for Cost-Effective System-Level Protection in Multi-core SoCs.", 2015 Euromicro Conference on Digital System Design, DSD 2015, Madeira, Portugal, August 26-28, 2015, pp. 41-48

[5] G. Kornaros, M. D. Grammatikakis, M. Coppola, "Towards Full Virtualization of Heterogeneous NoC-based Multicore Embedded Architectures", 10th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (IEEE/IFIP EUC 2012), pp. 345-352

[6] G. Kornaros and M. Coppola, "Enabling Efficient Job Dispatching in Accelerator-extended Heterogeneous Systems with Unified Address Space", Procs of 30th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2018, September 24-27, 2018

[7] G. Kornaros, "High-Speed Hardware Arbitration Supporting Priorities and Bounded Service Latency," IEEE Embedded Systems Letters, vol. 5, issue 2, pp. 21-24, Mar. 2013

[8] George Kornaros, "A Soft Multi-core Architecture for Edge Detection and DataAnalysis of Microarray Images", J. Syst. Architect. JSA (2009), Volume 56, Issue 1, January 2010, pp. 48-62

[9] G. Kornaros and M. Pratikakis, "VWQS: A dispatching mechanism of variable-size tasks in heterogeneous systems", International Conference on High Performance Computing & Simulation, HPCS 2016, Innsbruck, Austria, July 18-22, 2016, pp. 196-203

[10] G.Kornaros, "Application Specific Customizable Embedded Systems", Book Chapter 2, 39 pages, in "Multi-Core Embedded Systems", Eds. G.Kornaros, CRC Press/Taylor & Francis Group, Apr. 2010, ISBN: 978-1-4398-1161-0

[11] Joseph Amalraj1, Dr. J. John Raybin Jose,“A SURVEY PAPER ONCRYPTOGRAPHY TECHNIQUES”, IJCSMC, Vol. 5, Issue. 8, August 2016, pg.55–59.

[12] AdilJamil Zaru. Int. Journal of Engineering Research and Application www.ijera.com ISSN : 2248-9622, Vol.08, Issue 02, (Part -2) pp.68-71 2018

[13] Mansoor Ebrahim, Shujaat Khan, Umer Bin Khalid Symmetric Algorithm Survey: A Comparative Analysis International Journal of Computer Applications (0975 – 8887) Volume 61– No.20, January 2013

[14] Rob Stubbs An Overview of Symmetric Encryption and the Key Lifecycle https://www.cryptomathic.com/news-events/blog/an-overview-of-symmetric-encryption-and-the-key-lifecycle 2019

[15] Salah A. k. Albermany - Fatima RadiHamade, "Survey: Block cipher Methods" 2016

[16]Asaithambi.N, "A Study on Asymmetric Key Cryptography Algorithms", International Journal of Computer Science and Mobile Applications, Vol.3 Issue. 4,pg. 8-13 ©, IJCSMA ISSN: 2321-8363 2015

[17] Preneel B. Cryptographic Hash Functions: Theory and Practice. In: Soriano M., Qing S., López J. (eds) Information and Communications Security. ICICS 2010. Lecture Notes in Computer Science, vol 6476. Springer, Berlin, Heidelberg 2010

[18] Manoj Ranjan Mishra, Jayaprakash Kar A STUDY ON
DIFFIE-HELLMAN KEY EXCHANGE PROTOCOLS International Journal of Pure and Applied Mathematics Volume 114 No. 2 , 179-189 , 2017

[19] Qamar Jabeen, Fazlullah Khan, Muhammad Nouman Hayat, Haroon Koger M. Needham TEA, a Tiny Encryption Algorithm 1995

[20] Michael Barr Anthony Massa Programming Embedded Systems, Second Edition with C and GNU Development Tools , Book Chapter 1 Page 14 October 2006 ISBN: 9780596009830

[21] Pong P. Chu FPGA PROTOTYPING BY VHDL EXAMPLES Xilinx SpartanTM-3 Version Book Chapter 2 page 11 2008 ISBN 978-0-470-18531-5

[22] Michael Healy, Thomas Newe and Elfed Lewis Analysis of Hardware Encryption versus Software Encryption on Wireless Sensor Network Motes book Smart Sensors and Sensing Technology (pp.3-14) 2008

[23] David J. Wheeler & R ions on the Cache-coherent Zynq Platform",
ACM Trans. Reconfigurable Technol. Syst., Vol. 11, No. 4, Article 25, Publication date: January 2019. DOI: https://doi.org/10.1145/3277506

[24] ALEXANDER KROH and OLIVER DIESSEL, "Efficient Fine-grained Processor-logic Interact han, Syed Roohullah Jan, Farman Ullah A Survey : Embedded Systems Supporting By Different Operating Systems , Volume 2 | Issue 2 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099 2016

[25] J. Dahlstrom and S. Taylor. 2013. Migrating an OS scheduler into tightly coupled FPGA logic to increase attacker workload. In Proceedings of the IEEE Military Communications Conference (MILCOM'13). 986–991. DOI:https://doi.org/10.1109/MILCOM.2013.171

[26] Pratima Deshpande S Santhanalakshmi Lakshmi P Experimental Study of Diffie-Hellman Key Exchange Algorithm on Embedded Devices International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS-2017)

Παραρτήματα της Πτυχιακής Εργασίας

ΠΑΡΑΡΤΗΜΑ Α

Κώδικας εργασίας

Κώδικας για τον επεξεργαστή A90

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "platform.h"
#include "xil_printf.h"
#include "aes.h"
#include "xscutimer.h"

#define  debu
#define  JOB_POS 50
#define  DATA_SIZE  4
#define  ONE_TENTH 32500000
#define crypt

struct Job {
	int done; //0 1 2
	int exe;
	struct blck *DATApos;
	int priority;
	struct Job *next;
};

struct blck {
#ifdef crypt
	int pubMixA90;
#endif
	int free;  //0 1 2
```

```
32.         int DATA[DATA_SIZE][DATA_SIZE];
33.         int DATAB[DATA_SIZE][DATA_SIZE];
34.         int DATAC[DATA_SIZE][DATA_SIZE];
35. };
36.
37. struct signal {
38.         struct Job *cur;
39.         struct Job *head;
40. #ifdef crypt
41.         int pubMixA91;
42. #endif
43. };
44.
45. volatile struct blck *ram1 =
46.                 (volatile struct blck *) (XPAR_PS7_RAM_0_S_AXI_BASEADDR);
47. volatile struct signal *signalA91 = (volatile struct signal *) (0x40001FF0);
48.
49. #ifdef crypt
50. int a = 2;
51. int pub_mod=5;
52. int pub_base=23;
53. #endif
54.
55. int getpos() {
56.         int u;
57.         for (u = 0; u < JOB_POS; u++)
58.                 if ((ram1 + u)->free == 0) {
59.                         (ram1 + u)->free = 1;
60.                         return u;
61.                 }
62.         return -1;
63. }
64.
65. void initpos(int u) {
66.         for (int k = 0; k < DATA_SIZE; k++)
67.                 for (int f = 0; f < DATA_SIZE; f++) {
68.                         ram1[u].DATA[k][f] = u + 1;
69.                         ram1[u].DATAB[k][f] = u + 1;
70.                         ram1[u].DATAC[k][f] = 0;
71.                 }
72. #ifdef crypt
73.         WORD key_schedule[60], idx;
74.         BYTE enc_buf[128];
75.         BYTE key[1][32];
```

```
76.
77.     int a1 = a,A=1;
78.   //mix A
79.     while (a1--) {
80.                 A = A *pub_base;
81.           }
82.     ram1[u].pubMixA90 = A%pub_mod;
83.   //
84.     a1 = a;A=1;
85.     while (a1--) {
86.                     A = A * signalA91->pubMixA91;
87.                }
88.     printf("%d----\n",A%pub_mod);
89.     int mat[8] = { 0, 0, 0, 0, 0, 0, A%pub_mod };
90.     memcpy(key, mat, 32);
91.     aes_key_setup(key[0], key_schedule, 256);
92.
93.     for (idx = 0; idx < 4; idx++) {
94.           aes_encrypt(ram1[u].DATA[idx], enc_buf, key_schedule, 256);
95.           memcpy(ram1[u].DATA[idx], enc_buf, 16);
96.     }
97.
98.     for (idx = 0; idx < 4; idx++) {
99.           aes_encrypt(ram1[u].DATAB[idx], enc_buf, key_schedule, 256);
100.                 memcpy(ram1[u].DATAB[idx], enc_buf, 16);
101.             }
102.
103.       #endif
104.       }
105.
106.       void newnode(struct Job *ptr, int u) {
107.             ptr->next = malloc(sizeof(struct Job));
108.             ptr->next->priority = u + 1;
109.             ptr->next->DATApos = ram1 + u;
110.             ptr->next->done = 0;
111.             ptr->next->exe = -1;
112.             ptr->next->next = NULL;
113.       }
114.
115.       void printlist(struct Job *ptr) {
116.             printf("\nprint ============\n");
117.             while (ptr != NULL) {
118.
119.                   printf("\n===========\n");
```

```
120.
121.                 printf("\n %i  prio \n", ptr->priority);
122.                 printf("\n %i  done \n", ptr->done);
123.                 printf("\n %i  free \n", ptr->DATApos->free);
124.                 printf("\n %i  exe\n", ptr->exe);
125.                 printf(" \nA\n");
126.                 for (int y = 0; y < DATA_SIZE; y++) {
127.                       printf(" \n");
128.                       for (int s = 0; s < DATA_SIZE; s++)
129.                             printf("%i ", ptr->DATApos->DATA[y][s]);
130.                 }
131.
132.                 printf(" \nB\n");
133.
134.                 for (int y = 0; y < DATA_SIZE; y++) {
135.                       printf(" \n");
136.                       for (int s = 0; s < DATA_SIZE; s++)
137.                             printf("%i ", ptr->DATApos->DATAB[y][s]);
138.                 }
139.
140.                 printf(" \nC\n");
141.
142.                 for (int y = 0; y < DATA_SIZE; y++) {
143.                       printf(" \n");
144.                       for (int s = 0; s < DATA_SIZE; s++)
145.                             printf("%i ", ptr->DATApos->DATAC[y][s]);
146.                 }
147.                 ptr = ptr->next;
148.           }
149.           printf("\n============\n");
150.           printf(" \n");
151.     }
152.
153.     XScuTimer Timer;
154.     void print_hex(BYTE str[], int len) {
155.           int idx;
156.
157.           for (idx = 0; idx < len; idx++)
158.                 printf("%02x", str[idx]);
159.     }
160.
161.     int main() {
162.
163.           Xil_DCacheDisable();
```

```
164.
165.             printf("\nA90 start================\n");
166.          sleep(1);
167.
168.             struct Job *ptr;
169.
170.             int u = 0;
171.
172.             //TIMER INIT----------
173.
174.             unsigned long long time = 0, time2 = 0;
175.
176.             XScuTimer_Config *ConfigPtr;
177.             XScuTimer *TimerInstancePtr = &Timer;
   ConfigPtr = XScuTimer_LookupConfig(XPAR_PS7_SCUTIMER_0_DEVICE_ID);
178.             XScuTimer_CfgInitialize(TimerInstancePtr, ConfigPtr, ConfigPtr-
    >BaseAddr);
179.
180.             XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*50);
181.
182.             XScuTimer_Start(TimerInstancePtr);
183.
184.             //----------------------
185.             //first node
186.
187.             signalA91->head = malloc(sizeof(struct Job));
188.             signalA91->head->priority = -1;
189.             signalA91->head->next = NULL;
190.             signalA91->head->done = 1;
191.             signalA91->head->DATApos = (struct blck *) &ram1[JOB_POS];
192.             signalA91->head->exe = -1;
193.             signalA91->cur = signalA91->head;
194.             // init RAM
195.             for (int i = 0; i < JOB_POS; i++)
196.                 ram1[i].free = 0;
197.
198.             int g;
199.
200.             int cnt;
201.
202.             cnt = 0;
203.             //make jobs
204.             g = 5;
205.
```

```
206.            while (g--) {
207.
208.                printf("loop%i\n", g);
209.
210.                ptr = signalA91->head;
211.
212.                while (ptr->next != NULL)
213.                    ptr = ptr->next;
214.
215.                u = getpos();
216.
217.                printf("%i u\n", u);
218.
219.                initpos(u);
220.
221.                newnode(ptr, u);
222.
223.                cnt++;
224.            }
225.            printlist(signalA91->head);
226.            time = XScuTimer_GetCounterValue(TimerInstancePtr);
227.            //collect results
228.            while (1) {
229.
230.                ptr = signalA91->head;
231.
232.                if (!cnt)
233.                    break;
234.
235.                while (ptr != NULL) {
236.
237.                    if (ptr->DATApos->free == 2) {
238.                        cnt--;
239.                        ptr->DATApos->free = 0;
240.        #ifdef debug
241.                        printf("\n %i exe %i prior %i cnt\n", ptr->exe, ptr->priority,
242.                                cnt);
243.
244.                        printf("\nA\n");
245.
246.                        for (int y = 0; y < DATA_SIZE; y++) {
247.
248.                            printf(" \n");
249.
```

```
250.                            for (int s = 0; s < DATA_SIZE; s++)
251.                            printf("%i ", ptr->DATApos->DATA[y][s]);
252.                        }
253.
254.                        printf("\nB\n");
255.
256.                        for (int y = 0; y < DATA_SIZE; y++) {
257.
258.                            printf(" \n");
259.
260.                            for (int s = 0; s < DATA_SIZE; s++)
261.                            printf("%i ", ptr->DATApos->DATAB[y][s]);
262.                        }
263.
264.                        printf("\nC\n");
265.
266.                        for (int y = 0; y < DATA_SIZE; y++) {
267.
268.                            printf(" \n");
269.
270.                            for (int s = 0; s < DATA_SIZE; s++)
271.                            printf("%i ", ptr->DATApos->DATAC[y][s]);
272.                        }
273.        #endif
274.                    }
275.
276.                    //TODO delete node
277.                    ptr = ptr->next;
278.                }
279.            }
280.
281.            time2 = XScuTimer_GetCounterValue(TimerInstancePtr);
282.            printf("\n\n\n %llu end \n", time - time2);
283.
284.            return 0;
285.        }
286.
```

Κώδικας για τον επεξεργαστή A91

```
287.        #include <stdio.h>
```

```
288.    #include <stdlib.h>
289.    #include <math.h>
290.    #include "platform.h"
291.    #include "xil_printf.h"
292.    #include "aes.h"
293.    #include "xscutimer.h"
294.    #include "sha1.h"
295.
296.    #define ONE_TENTH 32500000
297.    #define  DATA_SIZE  4
298.    #define crypt
299.    #define time
300.
301.    struct Job {
302.        int done; //0 1 2
303.        int exe;
304.        struct blck *DATApos;
305.        int priority;
306.        struct Job *next;
307.    };
308.
309.    struct blck {
310.    #ifdef crypt
311.        int pubMixA90;
312.    #endif
313.        int free;   //0 1 2
314.        int DATA[DATA_SIZE][DATA_SIZE];
315.        int DATAB[DATA_SIZE][DATA_SIZE];
316.        int DATAC[DATA_SIZE][DATA_SIZE];
317.    };
318.
319.    struct signal {
320.        struct Job *cur;
321.        struct Job *head;
322.    #ifdef crypt
323.        int pubMixA91;
324.    #endif
325.    };
326.
327.    #ifdef crypt
328.    int b = 3;
329.    int pub_mod = 5;
330.    int pub_base = 23;
331.    #endif
```

```
332.
333.    void multiply(int mat1[][DATA_SIZE], int mat2[][DATA_SIZE],
334.            int res[][DATA_SIZE]) {
335.        int y = 0, i, j, k;
336.
337.        for (i = 0; i < DATA_SIZE; i++) {
338.            for (j = 0; j < DATA_SIZE; j++) {
339.                res[i][j] = 0;
340.                for (k = 0; k < DATA_SIZE; k++)
341.                    res[i][j] += mat1[i][k] * mat2[k][j];
342.            }
343.        }
344.    }
345.
346.    volatile struct blck *ram1 =
347.            (volatile struct blck *) (XPAR_PS7_RAM_0_S_AXI_BASEADDR);
348.    volatile struct signal *signalA91 = (volatile struct signal *) (0x40001FF0);
349.
350.    void printlist(struct Job *ptr) {
351.        printf("\nprint ============\n");
352.
353.        while (ptr != NULL) {
354.
355.            printf("\n===========\n");
356.
357.            printf("\n %i  prio \n", ptr->priority);
358.            printf("\n %i  done \n", ptr->done);
359.            printf("\n %i  free \n", ptr->DATApos->free);
360.            printf("\n %i  exe\n", ptr->exe);
361.            printf(" \nA\n");
362.            for (int y = 0; y < DATA_SIZE; y++) {
363.                printf(" \n");
364.                for (int s = 0; s < DATA_SIZE; s++)
365.                    printf("%i ", ptr->DATApos->DATA[y][s]);
366.            }
367.
368.            printf(" \nB\n");
369.
370.            for (int y = 0; y < DATA_SIZE; y++) {
371.                printf(" \n");
372.                for (int s = 0; s < DATA_SIZE; s++)
373.                    printf("%i ", ptr->DATApos->DATAB[y][s]);
374.            }
375.
```

```
376.                    printf(" \nC\n");
377.
378.                    for (int y = 0; y < DATA_SIZE; y++) {
379.                          printf(" \n");
380.                          for (int s = 0; s < DATA_SIZE; s++)
381.                                printf("%i ", ptr->DATApos->DATAC[y][s]);
382.                    }
383.                    ptr = ptr->next;
384.              }
385.              printf("\n============\n");
386.              printf(" \n");
387.        }
388.
389.        XScuTimer Timer;
390.
391.        int main() {
392.
393.              Xil_DCacheDisable();
394.              //TIMER INIT----------
395.              printf("\nA91 start=================\n");
396.
397.              printf("A91!!\n");
398.
399.              struct Job *tmp;
400.        #ifdef timer
401.              unsigned long long time = 0, time2 = 0;
402.              XScuTimer_Config *ConfigPtr;
403.              XScuTimer *TimerInstancePtr = &Timer;
404.              ConfigPtr =
    XScuTimer_LookupConfig(XPAR_PS7_SCUTIMER_0_DEVICE_ID);
405.              XScuTimer_CfgInitialize(TimerInstancePtr, ConfigPtr, ConfigPtr-
    >BaseAddr);
406.        #endif
407.
408.        #ifdef crypt
409.              int b1 = b, B = 1;
410.
411.              while (b1--) {
412.                    B = B * pub_base;
413.              }
414.              signalA91->pubMixA91 = B % pub_mod;
415.        #endif
416.              int f = 0;
417.              while (1) {
```

```
418.
419.                 tmp = signalA91->cur;
420.
421.                 if (tmp->done == 0 && tmp->DATApos->free == 1) {
422.
423.                     tmp->exe = ++f;
424.
425.                     tmp->done = 1;
426.         #ifdef crypt
427.                     B = 1;
428.                     b1 = b;
429.                     while (b1--) {
430.                         B = B * tmp->DATApos->pubMixA90;
431.                     }
432.                     printf("((%d}}", B % pub_mod);
433.
434.                     WORD key_schedule[60], idx;
435.                     BYTE enc_buf[128];
436.                     BYTE key[1][32];
437.
438.                     //
439.
440.                     int mat[8] = { 0, 0, 0, 0, 0, 0, B%pub_mod };
441.                     memcpy(key, mat, 32);
442.                     aes_key_setup(key[0], key_schedule, 256);
443.
444.                     for (idx = 0; idx < 4; idx++) {
445.                    aes_decrypt(tmp->DATApos->DATA[idx], enc_buf, key_schedule,
446.                                 256);
447.                         memcpy(tmp->DATApos->DATA[idx], enc_buf, 16);
448.                     }
449.
450.                     for (idx = 0; idx < 4; idx++) {
451.                   aes_decrypt(tmp->DATApos->DATAB[idx], enc_buf, key_schedule,
452.                                 256);
453.                         memcpy(tmp->DATApos->DATAB[idx], enc_buf, 16);
454.                     }
455.         #endif
456.         #ifdef timer
457.                     XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*50);
458.                     XScuTimer_Start(TimerInstancePtr);
459.                     time = XScuTimer_GetCounterValue(TimerInstancePtr);
460.         #endif
461.                     multiply(tmp->DATApos->DATA, tmp->DATApos->DATAB,
```

```
462.                                    tmp->DATApos->DATAC);
463.        #ifdef timer
464.                        time2 = XScuTimer_GetCounterValue(TimerInstancePtr);
465.                        XScuTimer_Stop(TimerInstancePtr);
466.                printf("\n\n\n %llu matrix %i\n", time - time2, tmp->priority);
467.        #endif
468.
469.                        tmp->priority = 0;
470.                        tmp->DATApos->free = 2;
471.                    }
472.                }
473.                return 0;
474.        }
475.
```

Κώδικας για τον επεξεργαστή MICROBLAZE

```
476.        #include <stdio.h>
477.        #define  JOB_POS 100
478.        #define  prior
479.        #define crypt
480.
481.        struct Job {
482.                int done; //0 1 2
483.                int exe;
484.                struct blck *DATApos;
485.                int priority;
486.                struct Job *next;
487.        };
488.
489.
490.        struct signal {
491.                struct Job *cur;
492.                struct Job *head;
493.                int pub_mod;
494.                int pub_base;
495.        #ifdef crypt
496.                int pubMixA91;
497.        #endif
498.        };
499.
500.        volatile struct signal *signalA91 = (volatile struct signal *) (0x40001FF0);
```

```
501.
502.    int main() {
503.
504.        disable_caches();
505.        Xil_DCacheDisable();
506.        struct Job *ptr, *tmp;
507.        int max_prio;
508.
509.    #ifdef prior
510.        while (1) {
511.            ptr = signalA91->head;
512.            max_prio = -1;
513.            while (ptr != NULL) {
514.                if (ptr->done == 0 && ptr->priority > max_prio) {
515.                    max_prio = ptr->priority;
516.                    tmp = ptr;
517.                }
518.                ptr = ptr->next;
519.            }
520.            signalA91->cur = tmp;
521.        }
522.    #else
523.        while (1) {
524.            ptr = signalA91->head;
525.
526.            while (ptr != NULL) {
527.                if (ptr->done == 0) {
528.                    signalA91->cur = ptr;
529.                }
530.                ptr = ptr->next;
531.            }
532.        }
533.
534.    #endif
535.
536.        return 0;
537.    }
538.
```