ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ - ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧ. Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΔΙΚΤΥΩΝ (ΤΜΗΥΤΔ)
ΕΡΓΑΣΤΗΡΙΟ ΗΛΕΚΤΡΟΝΙΚΩΝ ΑΙΣΘΗΤΗΡΩΝ

ΥΛΟΠΟΙΗΣΗ ΠΡΩΤΟΚΟΛΛΩΝ ΚΡΥΠΤΟΓΡΑΦΗΣΗΣ ΧΑΜΗΛΗΣ ΚΑΤΑΝΑΛΩΣΗΣ ΕΝΕΡΓΕΙΑΣ ΓΙΑ ΑΣΥΡΜΑΤΑ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Του

ΑΡΒΑΝΙΤΗ Γ. ΔΙΟΝΥΣΙΟΥ

Εκπονήθηκε υπό την επίβλεψη του Καθηγητή

Γεώργιου Σταμούλη

Εξεταστική Επιτροπή

Καθηγητής Τασιούλας Λέανδρος

Καθηγήτρια Χούστη Αικατερίνη

ΒΟΛΟΣ, ΙΟΥΝΙΟΣ 2008

ΕΥΧΑΡΙΣΤΙΕΣ

Για την εκπόνηση αυτής της εργασίας θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα Καθηγητή κ. Γεώργιο Σταμούλη για τη βοήθεια, τις συμβουλές και την υπομονή που επέδειξε σε όλο αυτό το διάστημα, καθώς και τον συνεπιβλέποντα Δρ. Παναγιώτη Κίκιρα. Η συμβολή του στην εκπόνηση και τελική μορφή της παρούσας μεταπτυχιακής εργασίας ήταν καθοριστική κάνοντας την όλη διαδικασία πολύ πιο ενδιαφέρουσα.

Ευχαριστίες επίσης, θα ήθελα να εκφράσω και στα υπόλοιπα μέλη της εξεταστικής επιτροπής της διπλωματικής εργασίας μου, την Καθηγήτρια Χούστη Αικατερίνη και τον Καθηγητή Τασιούλα Λέανδρο, για την προσεκτική ανάγνωση της εργασίας μου και για τις πολύτιμες υποδείξεις τους.

Διονύσιος Γ. Αρβανίτης

ΠΕΡΙΕΧΟΜΕΝΑ

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

Η σελίδα αυτή είναι σκόπιμα λευκή.

1. ΕΙΣΑΓΩΓΗ

1.1 ΑΝΤΙΚΕΙΜΕΝΟ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Σκοπός της παρούσας εργασίας είναι η υλοποίηση του πρωτοκόλλου ασφαλούς επικοινωνίας SecureMem/OTP (One Time Pad) για ασύρματα δίκτυα αισθητήρων, με χρήση του εργαλείου TinyOS, σε κώδικα NesC. Για την υλοποίηση του πρωτοκόλλου δημιουργήθηκε δίκτυο αποτελούμενο από κόμβους οι οποίοι κρυπτογραφούν τα δεδομένα τους με τον αλγόριθμο κρυπτογράφησης One Time Pad (OTP) και έναν σταθμό βάσης, ο οποίος αποκρυπτογραφεί τα δεδομένα που λαμβάνει από τους κόμβους του δικτύου.

Συμπεράσματα για την απόδοση του πρωτοκόλλου, ως προς την κατανάλωση ενέργειας, εξάγονται, κατόπιν σύγκρισης με το πρωτόκολλο TinySec του TinyOS και μιας υβριδικής έκδοσης αυτών OTP/TinySec, η οποία υλοποιήθηκε για τις ανάγκες της προσομοίωσης.

1.2 ΔΙΑΡΘΡΩΣΗ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Η εργασία αυτή, χωρίζεται σε δύο ενότητες. Στην πρώτη, καλύπτεται μέρος της θεωρίας για την ασφάλεια στα ασύρματα δίκτυα αισθητήρων και για το πρωτόκολλο που υλοποιήθηκε, ενώ στη δεύτερη καταγράφονται λεπτομέρειες για την εφαρμογή και τον τρόπο λειτουργίας της.

Ειδικότερα, στο πρώτο μέρος, γίνεται μια περιγραφή των βασικών αρχών ασφάλειας για τα ασύρματα δίκτυα αισθητήρων και αναλύονται διεξοδικά τα είδη των επιθέσεων και τα σημαντικότερα πρωτόκολλα ασφάλειας των δικτύων αυτού του τύπου (**Κεφάλαιο 2**).

Ακολουθεί το θεωρητικό κομμάτι (**Κεφάλαιο 3**), το οποίο περιγράφει τον αλγόριθμο κρυπτογράφησης One Time Pad, τις προϋποθέσεις χρήσης του και το μαθηματικό τρόπο με τον οποίο αποδεικνύεται η εγκυρότητα του αλγορίθμου. Το **Κεφάλαιο 4**, ασχολείται με τις Γεννήτριες Τυχαίων Αριθμών, που αποτελούν τη σημαντικότερη αρχή στην οποία στηρίζεται η απόδειξη του αλγορίθμου, ενώ το **Κεφάλαιο 5** περιγράφει αναλυτικά το πρωτόκολλο ασφάλειας SecureMem/OTP.

Το δεύτερο μέρος της εργασίας αφορά αποκλειστικά την υλοποίηση και απόδοση της εφαρμογής. Ειδικότερα, στο **Κεφάλαιο 6**, γίνεται μια σύντομη περιγραφή των εργαλείων που χρησιμοποιήθηκαν και για τον τρόπο με τον οποίο έγινε ο προγραμματισμός των αισθητήρων. Επιπλέον, γίνεται σύγκριση της απόδοσης, ως προς την κατανάλωση ενέργειας, του πρωτοκόλλου SecureMem/OTP σε σχέση με το πρωτόκολλο TinySec του TinyOS και μιας υβριδικής έκδοσης αυτών OTP/TinySec. Ολοκληρώνοντας, το **Κεφάλαιο 7**, αφιερώνεται στην εξαγωγή και διατύπωση συμπερασμάτων, ως προς την κατανάλωση ενέργειας, των πραναφερθέντων πρωτοκόλλων.

ΜΕΡΟΣ Ι - ΘΕΩΡΙΑ

Η σελίδα αυτή είναι σκόπιμα λευκή.

2. ΑΣΦΑΛΕΙΑ ΣΤΑ ΑΣΥΡΜΑΤΑ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ

2.1 ΕΙΣΑΓΩΓΗ- ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ

Στα αρχικά στάδιά της, η έρευνα γύρω από τα ασύρματα δίκτυα αισθητήρων, ήταν εστιασμένη στη διερεύνηση των θεμάτων που άπτονταν της κατανάλωσης ενέργειας στους κόμβους και ειδικότερα στην προσπάθεια για την ανάπτυξη κόμβων, αλγορίθμων και πρωτοκόλλων τέτοιων ώστε να ελαχιστοποιείται η κατανάλωση ενέργειας σε αυτούς. Σήμερα αν και το παραπάνω πρόβλημα δεν έχει αντιμετωπιστεί πλήρως, ένα νέο ερευνητικό πεδίο αναδύεται· αυτό το οποίο εξετάζει τα θέματα ασφάλειας στα ασύρματα δίκτυα αισθητήρων.

Η ασφάλεια είναι ο ακρογωνιαίος λίθος για μια σειρά από εφαρμογές στις οποίες δραστηριοποιούνται τα δίκτυα αυτού του τύπου, όπως για παράδειγμα σε βιοϊατρικές και στρατιωτικές εφαρμογές, σε εφαρμογές βιομηχανικού ελέγχου και γενικά οπουδήποτε, κρίσιμες αποφάσεις στρατηγικού επιπέδου, εξαρτώνται από πληροφορίες οι οποίες συγκεντρώνονται και επεξεργάζονται από αυτά.

Αν και η ασφάλεια δικτύων και υπολογιστικών συστημάτων είναι μια καλά εδραιωμένη επιστημονική περιοχή, με πρωτόκολλα και πρότυπα τα οποία τυγχάνουν ευρείας αναγνώρισης, η προσαρμογή και χρησιμοποίηση αυτών στα δίκτυα ασυρμάτων αισθητήρων, είναι τις περισσότερες φορές, αν όχι αδύνατη, πάρα πολύ δύσκολη, εξαιτίας των ιδιαίτερων χαρακτηριστικών των δικτύων αισθητήρων τόσο εξαιτίας των περιορισμών των κόμβων που τα απαρτίζουν (χαμηλή υπολογιστική ισχύς, περιορισμένες ικανότητες αποθήκευσης, περιορισμένη διαθέσιμη ενέργεια), όσο και εξαιτίας των ιδιαίτερων επιχειρησιακών χαρακτηριστικών των εφαρμογών στις οποίες χρησιμοποιούνται (λειτουργία σε αντίξοα -με κάθε μορφής «θόρυβο»- περιβάλλοντα, ελλιπής γνώση της τοπολογίας στην περιοχή ανάπτυξης, δυνατότητες αυτό-οργάνωσης και αυτόματης διόρθωσης δυσλειτουργιών, και τέλος λειτουργία χωρίς ανθρώπινη επιτήρηση στο μεγαλύτερο μέρος της ζωής τους).

2.2 Απειλες - Αρχες Ασφαλειας

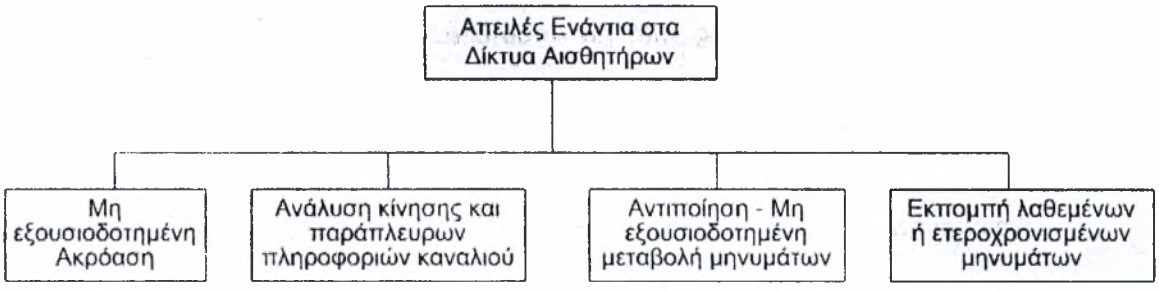
2.2.1 Απειλές

Οι κυριότερες απειλές ενάντια στην ασφαλή λειτουργία των δικτύων αισθητήρων είναι:

- **Μη εξουσιοδοτημένη Ακρόαση** - Eavesdropping: Ονομάζεται η απόκτηση πληροφοριών με μη ενεργητικούς τρόπους από ένα δίκτυο. Η φύση της επικοινωνίας στα δίκτυα αισθητήρων, η οποία βασίζεται στην χρησιμοποίηση ραδιοφωνικών συχνοτήτων για την διεξαγωγή της, την καθιστά ευπαθή σε υποκλοπές από οποιονδήποτε δέκτη βρίσκεται εντός της ακτίνας εκπομπής των κόμβων του δικτύου, ενώ σε κάποιες περιπτώσεις υπάρχει δυνατότητα υποκλοπών και από δέκτες οι οποίοι βρίσκονται μακριά από την ακτίνα εκπομπής των δικτύων, αρκεί οι υποκλοπείς να χρησιμοποιούν κεραίες υψηλού κέρδους. Το σημαντικότερο πρόβλημα που προκαλείται από την απειλή αυτή, σχετίζεται με τη δυνατότητα την οποία έχει ο υποκλοπέας να επεξεργάζεται τα μηνύματα που υποκλέπτει και στη συνέχεια, είτε να τα εκμεταλλεύεται ως έχουν, είτε να τα χρησιμοποιεί για την εκτόξευση εναντίων του δικτύου κάποιας από τις επιθέσεις που θα δούμε στη συνέχεια.
- **Εκπομπή λαθεμένων ή επανεκπομπή ετεροχρονισμένων παλαιοτέρων μηνυμάτων** (Message injection or replay attack): η εισαγωγή στο δίκτυο λαθεμένων ή παλαιών μηνυμάτων από μη εξουσιοδοτημένους κόμβους, είναι μια πάγια απειλή 1) ενάντια στους κόμβους διότι προκαλεί ταχεία αποφόρτισή τους, εξαιτίας της εντατικής χρήσης του πομποδέκτη τους, και 2) ενάντια στην αξιοπιστία του δικτύου εξαιτίας των λαθεμένων ή ετεροχρονισμένων πληροφοριών οι οποίες διακινούνται από αυτό. Τέτοιου είδους τακτικές είναι τυπικές στις επιθέσεις τύπου Denial Of Service.
- **Αντιποίηση** και μη εξουσιοδοτημένη μεταβολή των μηνυμάτων (Impersonation & message modification): Η αντιποίηση λαμβάνει χώρα όταν ένας μη εξουσιοδοτημένος κόμβος προσποιείται τη συμπεριφορά ενός κόμβου του δικτύου. Η μη εξουσιοδοτημένη μεταβολή ενός μηνύματος συμβαίνει όταν κάποιος αντίπαλος επιτύχει να συλλέξει ένα μήνυμα, να το μεταβάλει και στη συνέχεια να το επανεκπέμψει στο δίκτυο. Η παραπάνω διαδικασία είναι τυπική στις επιθέσεις τύπου man-in-the middle.

- **Ανάλυση κίνησης** και εκμετάλλευσης παράπλευρων πληροφοριών καναλιού (Traffic - Side channel analysis): Ανάλυση κίνησης συμβαίνει όταν ένας αντίπαλος είναι σε θέση να καταγράφει όλη την κίνηση σε μια περιοχή του δικτύου. Η ανάλυση της κίνησης μπορεί να αποδειχθεί ιδιαίτερα χρήσιμο εργαλείο στα χέρια ενός ικανού αντιπάλου, διότι μπορεί, με κατάλληλη επεξεργασία να του αποκαλύψει την τοπολογία του δικτύου, πρότυπα και συνήθειες επικοινωνίας, σχέσεις μεταξύ των στοιχείων του δικτύου, σημαντικού για τη λειτουργία του δικτύου κόμβους κ.λπ.

Η ανάλυση τύπου Side channel βασίζεται στις «παράπλευρες» πληροφορίες που μπορούν να εξαχθούν από αυτό. Ως παράπλευρη χαρακτηρίζεται οποιαδήποτε πληροφορία η οποία μπορεί να ανακτηθεί από τον κόμβο, και η οποία δεν είναι το ίδιο το μήνυμα, αλλά πληροφορία όπως απαιτούμενος χρόνος για την εκτέλεση κρυπτογραφικών αλγορίθμων, κατανάλωση ισχύος, μηνύματα λάθους κ.λπ.[1].



Σχήμα 2.1 Απειλές Ενάντια στα Δίκτυα Αισθητήρων.

Για να μπορεί να αντιμετωπίσει τις απειλές αυτές ένα σύστημα πρέπει να είναι σύμφωνο με τις αρχές ασφαλείας (αυθεντικοποίηση, ακεραιότητα δεδομένων, εμπιστευτικότητα κ.λπ.) οι οποίες παρουσιάζονται στη συνέχεια.

2.2.2 Αρχές Ασφαλείας

Για την αντιμετώπιση των απειλών όπως αυτές φαίνονται στο Σχήμα 2.1, ένα δίκτυο ασυρμάτων αισθητήρων πρέπει να μπορεί να ικανοποιεί τις αρχές της αυθεντικότητας, της ακεραιότητας, της εμπιστευτικότητας, της μη αποποίησης, της ελεγχόμενης πρόσβασης, και της φυσικής ασφάλειας. Οι παραπάνω αρχές ασφαλείας μπορούν να αναλυθούν ως εξής:

- ***Εμπιστευτικότητα***: Οι πληροφορίες οι οποίες διακινούνται στο δίκτυο, θα πρέπει να αποκαλύπτονται μόνο στους αποδέκτες στους οποίους απευθύνονται και μόνο σε αυτούς.

Οποιοσδήποτε άλλος δεν πρέπει ακόμα και εάν ένα μήνυμα καταλήξει σε αυτόν από λάθος, να είναι σε θέση να ανακτήσει τις πληροφορίες που περιέχονται σε αυτό.

- ***Μη Αποποίηση***: Η μη αποποίηση συνίσταται στην ικανότητα του δικτύου, να ταυτοποιεί μοναδικά οποιαδήποτε δραστηριότητα στο δίκτυο, ώστε να μπορεί να εντοπιστεί οποιαδήποτε μη επιθυμητή δραστηριότητα σε αυτό.
- ***Αυθεντικότητα***: Οι πληροφορίες οι οποίες διακινούνται στο δίκτυο θα πρέπει να προέρχονται από τα στοιχεία που το αποτελούν και από τυχόν τρίτες σαφώς καθορισμένες «οντότητες». Οποιοσδήποτε άλλος δεν θα πρέπει να είναι ικανός να ανακτήσει πληροφορίες από το δίκτυο, είτε με λαθραία ακρόαση των διακινούμενων μηνυμάτων, είτε μετά από καταγραφή και επεξεργασία αυτών, ή να το χρησιμοποιήσει για την αποστολή δικών του μηνυμάτων.
- ***Ακεραιότητα***: Το δίκτυο πρέπει να παρέχει εγγυήσεις τέτοιες ώστε να διασφαλίζεται η επικοινωνία μεταξύ των κόμβων, χωρίς να είναι εφικτή η μεταβολή τού μηνύματος κατά τη διάρκεια της δρομολόγησης, από τον αποστολέα στον παραλήπτη από κάποιον κακόβουλο τρίτο, ή από λάθος μετάδοσης, ενώ, εάν κάτι τέτοιο συμβεί, θα πρέπει ο αποδέκτης να είναι σε θέση να εντοπίσει τη μεταβολή αυτή.
- ***Διαθεσιμότητα***: Οι υπηρεσίες του δικτύου, θα πρέπει να είναι διαθέσιμες προς χρήση, από οποιοδήποτε εξουσιοδοτημένο χρήστη του, οποτεδήποτε εκείνος τις χρειαστεί.
- ***Ελεγχόμενη Πρόσβαση***: Η πρόσβαση σε υπηρεσίες και πόρους του δικτύου θα πρέπει να γίνεται με ελεγχόμενο τρόπο, έτσι ώστε κάθε στοιχείο του δικτύου να έχει πρόσβαση, μόνο στις υπηρεσίες και πόρους για τις οποίες του επιτρέπεται.
- ***Φρεσκάδα Διακινούμενων Πληροφοριών*** (data freshess): Το δίκτυο πρέπει να εξασφαλίζει τη φρεσκάδα, ως προς το χρόνο, των πληροφοριών που διακινούνται από αυτό. Με αυτό τον τρόπο εξασφαλίζεται η προστασία των στοιχείων του από τις επιθέσεις επανεκπομπής μηνυμάτων, καθώς επίσης και από την άσκοπη κατανάλωση ενεργειακών πόρων στους κόμβους από μηνύματα τα οποία για κάποιο λόγο κυκλοφορούν σε ατέρμονες βρόχους (nested loops).

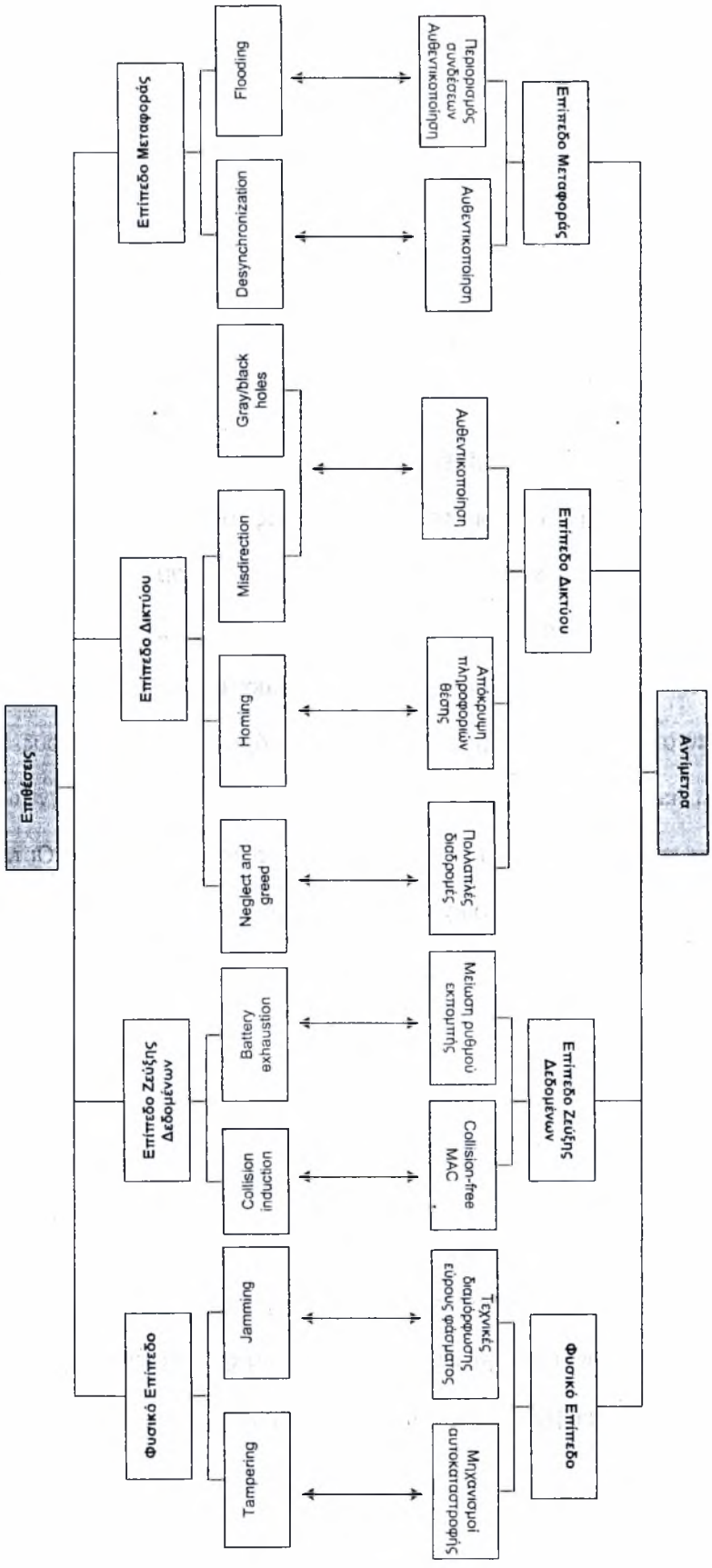
2.3 Επιθεσεις - Αντιμετρα

Στο Σχήμα 2.2 φαίνονται ταξινομημένες οι σημαντικότερες επιθέσεις και πιθανά αντίμετρα ενάντια σε αυτές ανά επίπεδο του OSI. Όπως είναι εμφανές, οι επιθέσεις αυτές στοχεύουν σε διαφορετικές λειτουργίες του δικτύου, γι' αυτό το λόγο τα αντίμετρα και η συνολική πολιτική ασφάλειας πρέπει να σχεδιάζονται όχι μεμονωμένα και ανεξάρτητα, αλλά με μια πολυεπίπεδη προοπτική και αρχιτεκτονική.

2.3.1 Επιθέσεις στο Φυσικό Επίπεδο

Οι επιθέσεις στο φυσικό επίπεδο μπορούν να στοχεύουν είτε στο μέσο μετάδοσης, είτε στον ίδιο τον κόμβο ως συσκευή. Στις πρώτες, ο επιτιθέμενος στοχεύει στην παύση λειτουργίας του δικτύου, μέσω της απαγόρευσης χρήσης σε αυτόν, του RF φάσματος εκπομπής (*jamming attack*). Η απαγόρευση επιτυγχάνεται με τη χρήση ισχυρών παραμβολών, οι οποίες κατακλύζουν την περιοχή του ραδιοφωνικού φάσματος συχνοτήτων που χρησιμοποιεί το δίκτυο με «θόρυβο», έτσι ώστε να είναι αδύνατη η ανταλλαγή οποιουδήποτε μηνύματος ανάμεσα στους κόμβους. Ένα αντίμετρο το οποίο αντιμετωπίζει ικανοποιητικά την παραπάνω απειλή είναι η χρήση τεχνικών διαμόρφωσης εύρους φάσματος στο σήμα και διαπηδούσης συχνότητας για τη μετάδοση [2]. Οι τεχνικές αυτές επιτρέπουν την επικοινωνία μόνο σε κόμβους οι οποίοι γνωρίζουν εκ των προτέρων το χρησιμοποιούμενο μοντέλο επικοινωνίας, αποκρύπτοντας την ύπαρξη και λειτουργία των κόμβων στην περιοχή ανάπτυξής τους.

Στις επιθέσεις ενάντια στους κόμβους ως συσκευές (*tampering attack*), ένας επιτιθέμενος εκμεταλλεύεται τη λειτουργία των κόμβων σε περιοχές μη ελεγχόμενες από τους φίλιους, και στοχεύει με την ανάκτησή τους, πέραν από τη φυσική καταστροφή τους, στην απόκτηση τυχόν πολύτιμων δεδομένων (π.χ. κρυπτογραφικό υλικό), τα οποία είναι αποθηκευμένα σε αυτούς. Η κύρια προστασία ενάντια σε αυτού του είδους την επίθεση, είναι η χρήση μηχανισμών προστασίας ενάντια στην παραβίαση (ενεργητικών ή παθητικών), καθώς και το μικρότερο δυνατό μέγεθος και η χρήση τεχνικών φυσικής απόκρυψης (χρώμα, σχήμα, λάμψη) των κόμβων.



Σχήμα 2.2 Επιθέσεις και Αντίμετρα στα Δίκτυα Αισθητήρων.

2.3.2 Επιθέσεις στο Επίπεδο Ζεύξης Δεδομένων

Στο επίπεδο της ζεύξης δεδομένων μπορούμε να εντοπίσουμε δύο είδη επιθέσεων: 1) τις επιθέσεις οι οποίες προκαλούν σύγκρουση των μεταδιδόμενων πακέτων δεδομένων, και 2) τις επιθέσεις οι οποίες στοχεύουν στην εξάντληση των αποθεμάτων των συσσωρευτών των κόμβων.

Οι πρώτες στοχεύουν σε δίκτυα τα οποία χρησιμοποιούν MAC πρωτόκολλα τα οποία λειτουργούν με το σχήμα *Ready-To-Send/Clear-To-Send (RTS/CTS)*. Στα δίκτυα αυτά ο επιτιθέμενος τοποθετεί στην περιοχή λειτουργίας τού δικτύου κόμβους οι οποίοι συλλαμβάνουν τα μηνύματα *(RTS/CTS)*, υποδύονται ότι είναι οι «νόμιμοι» αποδέκτες του αιτήματος και στην περίπτωση του *RTS* δεν αποστέλλουν ποτέ το *CTS* με αποτέλεσμα να εξαναγκάζεται ο κόμβος να εκπέμπει συνεχώς *RTS* πακέτα, καταργώντας ουσιαστικά τη δυνατότητα επικοινωνίας του με το υπόλοιπο δίκτυο [3]. Η βασικότερη μέθοδος αντιμετώπισης της παραπάνω επίθεσης, είναι με τη χρήση πρωτοκόλλων MAC τα οποία δεν επιτρέπουν τις συγκρούσεις πακέτων δεδομένων καθώς και η χρήση κωδίκων διόρθωσης λαθών.

Οι δεύτερες επιθέσεις στοχεύουν στην εξάντληση των ενεργειακών αποθεμάτων των κόμβων, την οποία επιτυγχάνουν, π.χ. αιτούμενα τη δρομολόγηση μεγάλων σε μέγεθος μηνυμάτων, με τη συνεχή χρήση του υποσυστήματος μετάδοσής τους, το οποίο, καταναλώνει την περισσότερη ενέργεια από όλα τα υποσυστήματα του κόμβου. Η κύρια άμυνα σε αυτού του είδους την επίθεση είναι η δρομολόγηση μηνυμάτων μόνο μετά από αυθεντικοποίηση του αποστολέα, η φραγή μηνυμάτων με μέγεθος μεγαλύτερο από το μέγεθος των τυπικών μηνυμάτων της εφαρμογής που εξυπηρετεί το δίκτυο.

2.3.3 Επιθέσεις στα Επίπεδα Δικτύου και Μεταφοράς

Στο επίπεδο δικτύου μπορούν να εκτοξευτούν μια σειρά από επιθέσεις οι οποίες σχετίζονται με την εκμετάλλευση αδυναμιών των πρωτοκόλλων δρομολόγησης. Έτσι ένας επιτιθέμενος μπορεί να συμμετέχει στην αλυσίδα δρομολόγησης των μηνυμάτων και να καταστρέφει όσα πακέτα φτάνουν σε αυτόν (*black hole attacks*), ή όσα μηνύματα λαμβάνει να τα κρατά και να τα επανεκπέμπει ετεροχρονισμένα (*grey hole attacks*) ή να τα προωθεί σε λαθεμένες κατευθύνσεις δημιουργώντας ατέρμονες βρόχους (*misdirections attacks*), προκαλώντας με αυτό τον τρόπο, σε όσους κόμβους συμμετέχουν στη δρομολόγηση, αναίτια κατανάλωση ενέργειας [4].

Μια επιπλέον κατηγορία επιθέσεων στο επίπεδο δικτύου είναι η επίθεση τύπου *homing*. Η επίθεση αυτή βασίζεται στην ανάλυση της κίνησης του δικτύου με σκοπό τον εντοπισμό κρίσιμων για τη λειτουργία του δικτύου κόμβων (π.χ. επικεφαλής ομάδων, διαχειριστές κρυπτογραφικών κλειδιών), τους οποίους προσπαθεί στη συνέχεια με κάποια άλλου τύπου επίθεση να εξουδετερώσει.

Η καλύτερη άμυνα ενάντια σε αυτού του είδους τις επιθέσεις στηρίζεται σε πρωτόκολλα δρομολόγησης, τα οποία παρακολουθούν συνεχώς τη λειτουργία του δικτύου, εντοπίζουν μη φυσιολογικές συμπεριφορές και τις απομονώνουν, και είναι σε θέση να παρέχουν εναλλακτικά μονοπάτια δρομολόγησης ώστε να αποφεύγονται οι περιοχές στις οποίες δρουν οι επιτιθέμενοι. Όμως η σημαντικότερη ενέργεια η οποία θωρακίζει το δίκτυο απέναντι σε αυτού του είδους τις επιθέσεις είναι η συμμετοχή στη δρομολόγηση μόνο κατάλληλα εξουσιοδοτημένων κόμβων.

2.4 Κατηγοριοποιηση των Πρωτοκολλων Ασφαλειας

2.4.1 Κατηγοριοποίηση των Πρωτοκόλλων Ασφαλείας

Οι βασικές κατηγορίες στις οποίες διακρίνονται τα πρωτόκολλα ασφάλειας των δικτύων αισθητήρων φαίνονται στο Σχήμα 2.3.

Σχήμα 2.3 Κατηγοριοποίηση Πρωτοκόλλων Ασφαλείας.

Η πρώτη κατηγορία αντιστοιχεί στα πρωτόκολλα τα οποία για την ανταλλαγή μηνυμάτων απαιτούν στο δίκτυο την ύπαρξη μιας τρίτης έμπιστης οντότητας, η οποία είτε παίζει το ρόλο του διανομέα κλειδιών ανάμεσα στους κόμβους, κάθε φορά που πρέπει να επικοινωνήσουν, είτε αναλαμβάνει με κάποιο τρόπο να πιστοποιεί και τους δύο κόμβους και να ενημερώνει τον καθένα για την πιστοποίηση του άλλου. Η μέθοδος αυτή εμφανίζει αρκετά προβλήματα, ιδιαίτερα στα δίκτυα με

μεγάλη κάλυψη στο χώρο, με σημαντικότερο από αυτό την αναγκαιότητα ύπαρξης της έμπιστης οντότητας στο δίκτυο τοποθετημένης στο χώρο έτσι ώστε να είναι προσπελάσιμη, συνεχώς και κατευθείαν από οποιονδήποτε κόμβο ανήκει σε αυτά. Το παραπάνω, σε συνδυασμό με τον ad-hoc τρόπο ανάπτυξης και κατανομής της τοπολογίας του δικτύου στο χώρο, καθιστά τα πρωτόκολλα αυτής της κατηγορίας μη χρησιμοποιήσιμα για τις περισσότερες εφαρμογές στις οποίες χρησιμοποιούνται τα ασύρματων δίκτυα αισθητήρων.

Η δεύτερη κατηγορία πρωτοκόλλων περιλαμβάνει εκείνα τα οποία χρησιμοποιούν για τη διεξαγωγή της επικοινωνίας κρυπτογραφία δημόσιου-ιδιωτικού κλειδιού. Όμως αποδεικνύεται ότι η μέθοδος αυτή δεν είναι κατάλληλη για χρήση στα δίκτυα αισθητήρων εξαιτίας της μεγάλης κατανάλωσης ενέργειας για την εκτέλεση των απαιτουμένων από το πρωτόκολλο βημάτων (διαδικασία hand-shaking, ανταλλαγή κλειδιών), αλλά και της πολυπλοκότητας στη διαχείριση της απαιτούμενης υποδομής της.

Τέλος, η τρίτη κατηγορία, περιλαμβάνει τα πρωτόκολλα τα οποία χρησιμοποιούν για την επικοινωνία, τεχνικές συμμετρικής κρυπτογραφίας, με κλειδιά τα οποία έχουν προ-υπολογιστεί και τοποθετηθεί στους κόμβους, πριν από την ανάπτυξη του δικτύου. Οι επικοινωνία ανάμεσα στους κόμβους γίνεται με τη χρησιμοποίηση των κλειδιών που βρίσκονται ήδη στη διάθεση τους, με γνωστό σε όλους τους κόμβους τρόπο επιλογής.

2.4.2 Επισκόπηση Διαθέσιμης Βιβλιογραφίας

Στις επόμενες παραγράφους, θα προχωρήσουμε σε μια συνοπτική επισκόπηση της διαθέσιμης βιβλιογραφίας γύρω από την ασφάλεια στα δίκτυα αισθητήρων. Οι **Karlof** και **Wagner** αναλύουν στο [5] τις επιθέσεις στο επίπεδο δρομολόγησης δεδομένων, για μια σειρά από δημοφιλή πρωτόκολλα των δικτύων αισθητήρων και προτείνουν μια σειρά από αντίμετρα για την αντιμετώπισή τους. Παρομοίως, οι **Wood** και **Stankovic** παρέχουν στο [6] μια επισκόπηση σε μια ποικιλία από DoS επιθέσεις και προτείνουν πιθανά αντίμετρα και αμυντικές τεχνικές ενάντια σε αυτές. Ο **Carman** αναλύει στο [7] τις υπάρχουσες τεχνολογίες διαχείρισης κρυπτογραφικών κλειδιών και τις απαιτήσεις που έχουν αυτές σε κατανάλωση ενέργειας. Ειδικότερα, η ανάλυσή του εστιάσθηκε στους απαιτούμενους υπολογιστικούς πόρους και στην επίπτωση των διαδικασιών ασφαλείας σε μια ποικιλία από πλατφόρμες οι οποίες χρησιμοποιούνται στα δίκτυα αισθητήρων. Ο **Potlapally** παρουσίασε στο [8] μια διεξοδική ανάλυση

των απαιτήσεων σε ενέργεια ενός μεγάλου αριθμού αλγορίθμων κρυπτογράφησης, από εκείνους που χρησιμοποιούνται από υφιστάμενα πρωτόκολλα ασφαλείας των δικτύων αισθητήρων.

Στο "*Secure Pebblenets*" [9], ο **Basagni** πρότεινε το δίκτυο να προστατεύεται από ένα μόνο κλειδί, το οποίο να αλλάζει ανά τακτά χρονικά διαστήματα. Πριν από κάθε αλλαγή του κλειδιού, επιλέγεται ο κόμβος με τη μεγαλύτερη διαθέσιμη ενέργεια, ο οποίος και αναλαμβάνει τη δημιουργία του νέου κλειδιού, το οποίο διανέμεται σε όλους τους κόμβους διαμέσου ενός ενεργειακά αποτελεσματικού αλγόριθμου. Ο **Zhu** παρουσίασε στο [10] το πρωτόκολλο LEAP (Localized Encryption and Authentication Protocol), το οποίο χρησιμοποιεί τέσσερα είδη κλειδιών σε κάθε κόμβο, για τέσσερις διαφορετικές λειτουργίες. Σε αυτά περιλαμβάνονται ένα ατομικό κλειδί για κάθε κόμβο το οποίο το γνωρίζει και ο σταθμός βάσης, και ένα ομαδικό κλειδί το οποίο το γνωρίζουν όλοι οι κόμβοι του δικτύου. Ο **Perrig** ανέπτυξε την αρχιτεκτονική ασφαλείας SPINS, η οποία αποτελείται από μια σειρά από πρωτόκολλα τα οποία στηρίζονται, για την παροχή ασφάλειας και αυθεντικοποιημένης εκπομπής, στις αρχές της συμμετρικής κρυπτογραφίας. Η αρχιτεκτονική SPINS στηρίζεται σε δύο δομικά στοιχεία, το SNEP, ένα πρωτόκολλο το οποίο παρέχει εμπιστευτικότητα, αυθεντικοποίηση και φρεσκάδα των δεδομένων, και το μTESLA, ένα πρωτόκολλο το οποίο εξασφαλίζει την αυθεντικότητα των εκπομπών [11]. Η βασική ιδέα πίσω από την αρχιτεκτονική SPINS, βασίζεται στην ιδέα ότι κάθε κόμβος μοιράζεται ένα μυστικό κλειδί, τη γνώση του οποίου μοιράζεται με έναν έμπιστο σταθμό βάσης, ο οποίος είναι διαρκώς διαθέσιμος και ικανός να επικοινωνεί με οποιονδήποτε κόμβο. Όπου και αν αυτός βρίσκεται στο δίκτυο. Όταν δύο κόμβοι απαιτείται να επικοινωνήσουν μεταξύ τους με ασφάλεια, ο σταθμός βάσης αναλαμβάνει να παρεμβληθεί ανάμεσα τους και να λειτουργήσει ως κέντρο διανομής κρυπτογραφικών κλειδιών.

Το 2004, οι **Karlof** et al. [12], παρουσίασαν την αρχιτεκτονική ασφάλειας TinySec, η οποία συμπεριλαμβάνεται στο TinyOS. Βασίζεται στην ιδέα ότι ο εντοπισμός μη-εξουσιοδοτημένων πακέτων είναι πιο εύκολος και επιθυμητός σε επίπεδο ζεύξης δεδομένων. Με αυτό τον τρόπο, μπορεί να αποφευχθεί η άσκοπη δρομολόγηση πακέτων, εξοικονομώντας ενέργεια και bandwidth. Παρέχει μηχανισμούς αυθεντικοποίησης και ελέγχου ακεραιότητας των μηνυμάτων (με χρήση Message Authentication Code, CBC-MAC), εμπιστευτικότητα (μέσω κρυπτογράφησης SkipJack), σημασιολογική ασφάλεια (με χρήση Initialization Vectors, IV) και προστασία από επανάληψη μηνυμάτων.

Μία σημαντική εργασία στο χώρο της ασφάλειας των δικτύων αισθητήρων είναι αυτή των **Eschenauer** και **Gligor**, οι οποίοι ήταν οι πρώτοι που παρουσίασαν ένα σύστημα ασφαλούς λειτουργίας δικτύου ασυρμάτων αισθητήρων το οποίο βασιζόταν στην ύπαρξη μιας δεξαμενής από κρυπτογραφικά κλειδιά τα οποία υπολογίζονταν και τοποθετούνταν στους κόμβους πριν από την ανάπτυξή τους στην περιοχή παρατήρησης [13], εξασφαλίζοντας έτσι με αυτό τον τρόπο, την ασφαλή διανομή, ανανέωση και κατάργηση των κρυπτογραφικών κλειδιών καταναλώνοντας όσο το δυνατόν λιγότερη ενέργεια, εξαιτίας της σημαντικής μείωσης τόσο των απαιτούμενων υπολογισμών (τα κρυπτογραφικά κλειδιά ήταν είδη υπολογισμένα), όσο και της αναγκαιότητας ανταλλαγής μηνυμάτων. Οι παραπάνω βασικές ιδέες των Eschenauer και Gligor, βελτιώθηκαν και συμπληρώθηκαν και πλέον μπορεί κάποιος να βρει πολλές αναφορές [14-16] στη βιβλιογραφία οι οποίες βασίζονται σε αυτές.

Πίνακας 2.1 Χαρακτηριστικά των Πρωτοκόλλων Ασφαλείας.

Πρωτόκολλο	Κρυπτογράφηση	Χαρακτηριστικά	Μειονεκτήματα
Carman	ασύμμετρη	Βασίζεται σε κοινό ομαδικό κλειδί και στη δημιουργία κρυπτογραφημάτων με τη χρήση του ID του κόμβου	Η ασύμμετρη κρυπτογραφία δεν είναι ενεργειακά συμφέρουσα τεχνική
Perrig (SPINS) **Zhu (LEAP)** **Undercoffer**	συμμετρική	Απαιτεί την ύπαρξη τρίτης έμπιστης οντότητας	Προβλήματα που σχετίζονται με την ύπαρξη της έμπιστης οντότητας
Eschenauer, Gligor **Chan** **Kikiras, Avaritsiotis**	συμμετρική	Βασίζονται στη διανομή πριν την ανάπτυξη του δικτύου των κρυπτογραφικών κλειδιών	Υψηλές απαιτήσεις μνήμης για την προαποθήκευση των κλειδιών, σε περίπτωση μεγάλου # κόμβων
Karlof et al. (TinySec)	συμμετρική	Αυθεντικοποίηση σε επίπεδο ζεύξης δεδομένων	Κάνει χρήση ενός καθολικού κλειδιού
Basagni (Pebblenets)	συμμετρική	Βασίζεται στην ύπαρξη κατάλληλων κλειδιών για κάθε λειτουργία	Ο τρόπος επιλογής του κατάλληλου κόμβου απαιτεί την ανταλλαγή μεγάλου αριθμού μηνυμάτων

Η σελίδα αυτή είναι σκόπιμα λευκή.

3. ΑΛΓΟΡΙΘΜΟΣ ΚΡΥΠΤΟΓΡΑΦΗΣΗΣ ONE-TIME PAD

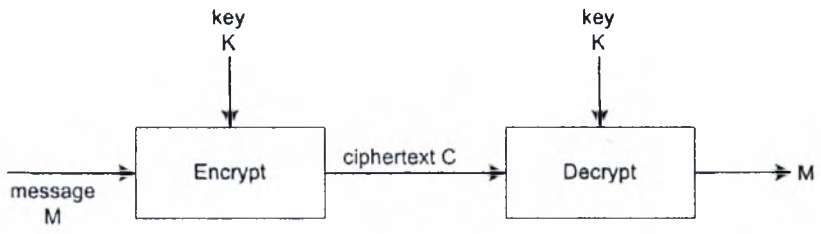
3.1 ΕΙΣΑΓΩΓΗ

Η επιλογή ενός αλγόριθμου κρυπτογράφησης για τα ασύρματα δίκτυα αισθητήρων εξαρτάται, κατά κύριο λόγο, από την ενέργεια που καταναλώνει το υποσύστημα επεξεργασίας του κόμβου στην ενεργή περιοχή λειτουργίας του, τη συχνότητας λειτουργίας του, και τον αριθμό των κύκλων που απαιτούνται από αυτό για τον υπολογισμό του αλγορίθμου. Για το λόγο αυτό, επιλέχθηκε ο αλγόριθμος One-Time Pad, ο οποίος διακρίνεται για τη χαμηλή κατανάλωση ενέργειας που τον χαρακτηρίζει λόγω απλότητας.

3.2 ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΟΥ

Ο αλγόριθμος *OTP (One-Time Pad)* προτάθηκε για πρώτη φορά το 1917 από τον *G. Vernam* [17], ως μέθοδος για την κρυπτογράφηση ενσύρματων και ασύρματων επικοινωνιών, ενώ το 1940 ο *C. Shannon* [18] απέδειξε ότι ο αλγόριθμος είναι απολύτως ασφαλής και δεν υπάρχει τρόπος να σπάσει εάν τηρηθούν ορισμένες προϋποθέσεις (§3.4).

Η ιδέα πίσω από τον OTP είναι η δημιουργία μιας τυχαίας σειράς από αλφαριθμητικούς χαρακτήρες, η οποία θα παίξει τον ρόλο του κλειδιού (Pad), ίσου τουλάχιστον μεγέθους με το μήνυμα το οποίο προορίζεται για κρυπτογράφηση και στη συνέχεια, η εκτέλεση της λογικής πράξης XOR ανάμεσα στο κλειδί και στο μήνυμα για την παραγωγή του κρυπτογραφήματος. Για να γίνει περισσότερο προφανής η παραπάνω διαδικασία, θα παρουσιάσουμε ένα απλό παράδειγμα:



Εικόνα 3.1 Αποστολή μηνύματος στο δίκτυο.

Έστω ένα μήνυμα *M* τέτοιο ώστε M=011010 και έστω K=101100 το κλειδί (Pad) το οποίο έχει δημιουργηθεί με τυχαίο τρόπο. Η εφαρμογή του λογικού τελεστή XOR σε αυτά τα δύο (Πίνακας 3.1) θα έδινε το κρυπτογράφημα C ώστε:

$$C = M \otimes K \Leftrightarrow C = 011010 \otimes 101100 \Leftrightarrow C = 110110 \tag{3.1}$$

Πίνακας 3.1 Πίνακας Τιμών XOR.

XOR	Y=0	Y=1
X=0	0	1
X=1	1	0

Η αποκρυπτογράφηση του κρυπτογραφήματος είναι εξίσου απλή. Αρκεί ο αποδέκτης να έχει στην κατοχή του το κλειδί *K*, το οποίο χρησιμοποιήθηκε για την κρυπτογράφηση του *M*, διότι για τον λογικό τελεστή XOR ισχύει $X \otimes (Y \otimes Z) = (X \otimes Y) \otimes Z$, οπότε στην περίπτωση του παραδείγματός μας έχουμε:

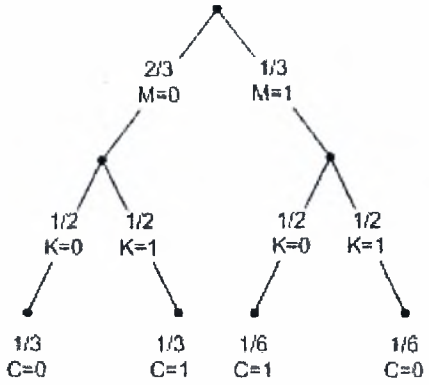
$$C = M \otimes K \Leftrightarrow M = C \otimes K \Leftrightarrow M = 110110 \otimes 101100 \Leftrightarrow M = 011010 \tag{3.2}$$

3.3 Αποδειξη Αλγοριθμου

Έστω μήνυμα M που μπορεί να πάρει τιμές 0,1. Έστω επίσης η πιθανότητα $P(M = 0) = 2/3$ και $P(M = 1) = 1/3$. Αν κάποιος λαθραναγνώστης διαβάσει το κρυπτογράφημα C=1, η πιθανότητα γίνεται πλέον δεσμευμένη, έστω

$$P(M = 0 | C = 1) = \frac{P(M = 0 \cap C = 1)}{P(C = 1)} = \frac{1/3}{1/2} = \frac{2}{3} \tag{3.3}$$

Παρατηρούμε ότι η υπό συνθήκη πιθανότητα του M=0, δεδομένου του C=1, είναι ίδια με την εκ των προτέρων (a priori) πιθανότητα του M=0.



Σχήμα 3.1 Δέντρο Πιθανοτήτων για τα Κρυπτογραφήματα.

Αυτό σημαίνει ότι ο λαθραναγνώστης δεν έχει μάθει τίποτα για το μήνυμα M από το κρυπτογράφημα C. Τυπικότερα ισχύει το παρακάτω:

Θεώρημα: Έστω τυχαίο μήνυμα $M \in \mathcal{M} = \{0,1\}^n$ και κλειδί $K \in \mathcal{K} = \{0,1\}^n$, όπου *n* είναι το μήκος του μηνύματος. Έστω επίσης, $C \in \mathcal{C}$ κρυπτογράφημα του M για το οποίο ισχύει $C = E_K(M)$. Για κάθε $m \in \mathcal{M}$ και $c \in \mathcal{C}$, ένα κρυπτογραφικό σύστημα είναι απόλυτα ασφαλές αν

$$P(M = m|C = c) = P(M = m) \tag{3.4}$$

Απόδειξη: Για κάθε $m \in \mathcal{M}$ και $c \in \mathcal{C}$ μήκους n-bits ισχύει το θεώρημα του Bayes κατά το οποίο:

$$P(M = m|C = c) = \frac{P(M = m \cap C = c)}{P(C = c)} \tag{3.5}$$

Επίσης, έχουμε

$P(M = m \cap C = c) = P(M = m) \cdot P(C = c|M = m)$ (εξ ορισμού πιθανότητας υπό συνθήκη)

$$= P(M = m) \cdot P(K = m \otimes c) \quad \textit{(K ανεξάρτητο από M)} \tag{3.6}$$

$$= P(M = m) \cdot 2^{-n} \quad \textit{(K επιλεγμένο ομοιόμορφα στο } \{0,1\}^n\textit{)}$$

και

$$P(C = c) = \sum_{\tilde{m} \in M} [P(M = \tilde{m})P(C = c \cap M = \tilde{m})] = \sum_{\tilde{m} \in M} [P(M = \tilde{m}) \cdot 2^{-n}] = 2^{-n} \tag{3.7}$$

Άρα

$$P(M = m|C = c) = \frac{P(M = m) \cdot 2^{-n}}{2^{-n}} = P(M = m) \tag{3.8}$$

□

3.4 Προϋποθεσεις Χρησης

Ωστόσο, αν και ο αλγόριθμος OTP είναι απλός, γρήγορος, και «αδιάσπαστος» δεν χρησιμοποιείται ευρέως. Οι βασικοί λόγοι για τους οποίους δεν έχει γνωρίσει τη χρήση που του αρμόζει είναι εξαιτίας των προϋποθέσεων που πρέπει να ικανοποιούνται ώστε να είναι πραγματικά «αδιάσπαστος». Αυτές είναι οι εξής:

1. Η διανομή των κλειδιών να έχει γίνει σε όλους τους κόμβους με ασφαλή τρόπο.
2. Για κάθε μήνυμα να χρησιμοποιείται ένα μοναδικό κλειδί, το οποίο στη συνέχεια να μη χρησιμοποιείται ποτέ ξανά (απόδειξη §3.4.1).
3. Το μέγεθος του κλειδιού να είναι, τουλάχιστον ίσο με το μέγεθος του κάθε μηνύματος που διακινείται (απόδειξη §3.4.2).
4. Η δημιουργία του κλειδιού να έχει γίνει με πραγματικά τυχαίο κρυπτογραφικά τρόπο (βλέπε Κεφάλαιο 4).

Όπως θα δούμε και στο επόμενο κεφάλαιο, το πρωτόκολλο SecureMem/OTP, έχει βρει λύσεις για τα παραπάνω προβλήματα εξουδετερώνοντας τις αρνητικές συνέπειές τους, ώστε να μπορεί να εκμεταλλεύεται το βασικότερο πλεονέκτημα του αλγορίθμου, το οποίο είναι η υπολογιστική του απλότητα και η ελάχιστη κατανάλωση ενέργειας που απαιτείται για την εκτέλεση του.

3.4.1 Two-Time Pad

Είδαμε προηγουμένως ότι κάθε κλειδί K πρέπει να χρησιμοποιείται μόνο μια φορά. Στην περίπτωση που ξαναχρησιμοποιηθεί τίθεται σε κίνδυνο η ασφάλεια του αλγορίθμου. Για παράδειγμα, έστω M_1 και M_2 κρυπτογραφημένα με το ίδιο κλειδί K, παράγοντας κρυπτογραφήματα C_1 και C_2 αντίστοιχα:

$$C_1 = M_1 \otimes K$$

$$C_2 = M_2 \otimes K$$

$$C_1 \otimes C_2 = (M_1 \otimes K) \otimes (M_2 \otimes K)$$

$$= (M_1 \otimes M_2) \otimes (K \otimes K)$$

$$= (M_1 \otimes M_2)$$

3.4.2 Μήκος Κλειδιού k

Θεώρημα: Για κάθε απολύτως ασφαλές σύστημα, για το οποίο τα επικοινωνούντα άκρα μοιράζονται κοινό κλειδί $K \in \mathcal{K}$ με το οποίο κρυπτογραφείται μήνυμα $M \in \mathcal{M}$, πρέπει να ισχύει $|\mathcal{K}| \geq |\mathcal{M}|$. Αν τα κλειδιά και τα μηνύματα είναι δυαδικές ακολουθίες $\mathcal{K} = \{0,1\}^n$ και $\mathcal{M} = \{0,1\}^l$, αυτό σημαίνει ότι ένα σύστημα είναι ασφαλές αν και μόνο αν $n \geq l$.

Απόδειξη: Έστω κρυπτογράφημα $C \in \mathcal{C}$. Έστω, επίσης, A το πλήθος των μηνυμάτων m, που μπορούν να προκύψουν από την αποκρυπτογράφηση του C, με κάποιο κλειδί K. Κάθε κλειδί $k \in \mathcal{K}$, ανταποκρίνεται το πολύ σε ένα m, καθώς η αποκρυπτογράφηση του C μπορεί να γίνει το πολύ με έναν τρόπο για κάθε κλειδί k. Άρα, $A \leq |\mathcal{K}|$.

Έστω επίσης, $A = |\mathcal{M}|$, δηλ. κάθε $m \in \mathcal{M}$ μπορεί να παράγει κρυπτογράφημα $c \in \mathcal{C}$. Πράγματι, αν αυτό δεν ίσχυε για κάποιο m, τότε $P(M = m \mid C = c) = 0$. Δεδομένου όμως, ότι η a priori πιθανότητα $P(M = m) > 0$, η παραπάνω ισότητα έρχεται σε σύγκρουση με το θεώρημα ανεξαρτησίας Shannon για απόλυτη ασφάλεια βάσει του οποίου ισχύει

$$P(M = m \mid C = c) = P(M = m) \neq 0$$

Άρα με συνεπαγωγή των δύο υποθέσεων, αποδεικνύεται ότι

$$A = |\mathcal{M}| \leq |\mathcal{K}| \tag{3.9}$$

□

Η σελίδα αυτή είναι σκόπιμα λευκή.

4. ΓΕΝΝΗΤΡΙΕΣ ΤΥΧΑΙΩΝ ΑΡΙΘΜΩΝ

4.1 Εισαγωγη

Οι γεννήτριες τυχαίων αριθμών χρησιμοποιούνται σε ένα πλήθος εφαρμογών όπως τυχερά παιγνίδια, στατιστική, προσομειώσεις, κρυπτογραφία, κ.λπ. Στην κρυπτογραφία ειδικότερα, υπάρχουν πολλές εφαρμογές, οι οποίες εξαρτώνται από την τυχαιότητα, την έλλειψη προβλεψιμότητας και την αδυναμία αναπαραγωγής της ακολουθίας που παράγει μια γεννήτρια, όπως:

- ☐ One-time pads
- ☐ Key generation
- ☐ Random nonces (π.χ. MD5 digest σε συνθηματικά χρήστη)
- ☐ Initialization Vectors (IVs)
- ☐ Κατακερματισμός (hashing salt)

Σε αλγόριθμους κρυπτογράφησης όπως ο OTP, επιβάλλεται τα κλειδιά που χρησιμοποιούνται να είναι πραγματικά τυχαία και κατ' επέκταση μη προβλέψιμα. Σε τέτοιου είδους εφαρμογές, αν η επιλογή των κλειδιών δεν γίνει με σωστό τρόπο, τίθεται σε κίνδυνο η ασφάλεια όλου του συστήματος. Στη συνέχεια αυτού του κεφαλαίου γίνεται μια σύντομη επισκόπηση των μεθόδων παραγωγής τέτοιων αριθμών μεταξύ των οποίων και η μέθοδος του random.org [19] (βλέπε §4.3.5), η οποία χρησιμοποιήθηκε στην παρούσα εργασία.

4.2 Γεννητρια Τυχαιων Αριθμων

Γεννήτρια Τυχαίων Αριθμών (Random Number Generator, RNG) ορίζεται η υπολογιστική ή φυσική συσκευή που παράγει ακολουθίες αριθμών ή συμβόλων, οι οποίες στερούνται οποιοδήποτε πρότυπο, δηλ. εμφανίζονται κατά τυχαίο τρόπο.

Ένας τρόπος παραγωγής τυχαίων αριθμών, είναι με τη μέτρηση κάποιου φυσικού φαινομένου, το οποίο αναμένεται να είναι τυχαίο αντισταθμίζοντας στη συνέχεια τις πιθανές πολώσεις που εισάγονται από τη διαδικασία μέτρησης. Ο άλλος τρόπος χρησιμοποιεί ντετερμινιστικούς αλγόριθμους

που παράγουν ακολουθίες αριθμών. Ο πρώτος «τυχαίος» αριθμός αυτής της σειράς παράγεται βάσει μιας αρχικής τιμής που ονομάζεται σπόρος (seed). Ο πρώτος τύπος ονομάζεται γεννήτρια πραγματικά τυχαίων αριθμών (TRNG), ενώ ο τελευταίος καλείται γεννήτρια ψευδοτυχαίων αριθμών (PRNG) και δεν μπορεί να θεωρηθεί πραγματικά τυχαίος, καθώς το απότελεσμα που προκύπτει είναι προβλέψιμο (για τον ίδιο σπόρο παράγεται ίδια ακολουθία).

Απαραίτητη προϋπόθεση για να είναι μια ακολουθία αριθμών πραγματικά τυχαία είναι η τήρηση των εξής στατιστικών ιδιοτήτων:

- ***Ομοιομορφία*** (Uniformity). Καθορίζεται παράγοντας μη-πολωμένα σύνολα αριθμών. Για παράδειγμα, σε αριθμοσύνολο [0,999] δεν πρέπει να ανοίκουν στο σύνολο [0, 499] περισσότεροι από τους μισούς αριθμούς.
- ***Ανεξαρτησία*** (Independence). Δεν υπάρχει συσχετισμός μεταξύ των αριθμών. Με άλλα λόγια, είναι αδύνατον να προβλέψεις έναν αριθμό σε μια ακολουθία, δεδομένης της γνώσης των προηγουμένων. Η πιθανότητα δηλ. παρατήρησης ενός αριθμού είναι ανεξάρτητη από τους προηγούμενους.
- ***Άθροισμα*** (Summation). Το άθροισμα δύο διαδοχικών τυχαίων αριθμών είναι ισοπίθανα άρτιο ή περιττό.
- ***Επανάληψη*** (Duplication). Αριθμοί του συνόλου επαναλαμβάνονται στην ακολουθία, με τυχαίο τρόπο.

Η ποιότητα της τυχαιότητας μπορεί να μετρηθεί με διάφορους τρόπους. Ο συνηθέστερος είναι η μέτρηση της εντροπίας της ακολουθίας. Του μέτρου δηλ. της αταξίας ή της τυχαιότητας ενός συστήματος. Όσο μεγαλύτερη είναι αυτή, τόσο πιο δύσκολη είναι η πρόβλεψη κάποιου αριθμού με τη γνώση των προηγούμενων. Υπάρχουν διάφορες μαθηματικές τεχνικές για την εκτίμηση της εντροπίας μιας ακολουθίας συμβόλων. Καμμία όμως δεν είναι σε θέση να διακρίνει μια πραγματικά τυχαία από μια ψευδοτυχαία ακολουθία.

Τα πρότυπα RFC-4086 (http://www.ietf.org/rfc/rfc4086.txt), FIPS Pub 140-2 (Security Requirements for Cryptographic Modules) και NIST 800-22 (A Statistical Test Suite for RNG and PRNG) περιλαμβάνουν ένα πλήθος ελέγχων που μπορούν να χρησιμοποιηθούν για το σκοπό αυτό.

4.3 ΦΥΣΙΚΑ ΦΑΙΝΟΜΕΝΑ ΜΕ ΙΔΙΟΤΗΤΕΣ ΤΥΧΑΙΟΤΗΤΑΣ

Μια κλασική προσέγγιση για τη χρήση φυσικής τυχαιότητας είναι η μετατροπή μιας πηγής θορύβου σε ακολουθία τυχαίων bit, μέσω A/D μετατροπέα. Ο θόρυβος, στη συνέχεια, ενισχύεται, φιλτράρεται και με τη βοήθεια ενός Συγκριτή Τάσης παράγεται λογικό σήμα που μεταβάλλει την κατάσταση σε τυχαία διαστήματα. Κατά τη φάση της ενίσχυσης του θορύβου, πρέπει να λαμβάνεται μέριμνα προς αποφυγή ανεπιθύμητων σημάτων (όπως power line hum). Τέλος, η ακολουθία μετατρέπεται σε σήμα RS-232 για τη σύνδεσή του σε υπολογιστή μέσω σειριακής. Το λογισμικό αντιλαμβάνεται αυτές τις λογικές τιμές σαν χαρακτήρες "line noise" [1].

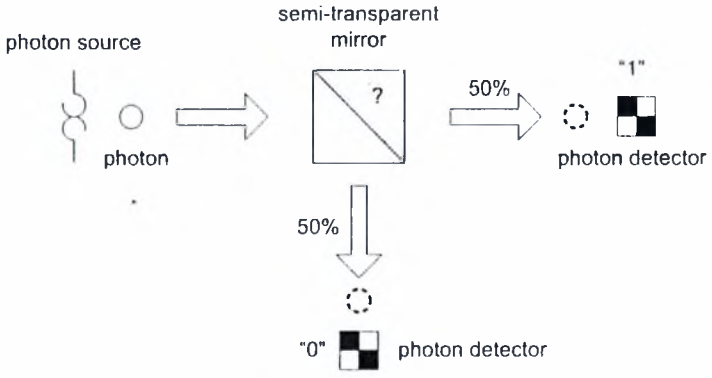
Κατά γενική ομολογία, αν υπάρχουν πραγματικά τυχαίοι αριθμοί, αυτοί μπορούν να εντοπιστούν σε φυσικές διαδικασίες, οι οποίες από όσο γνωρίζουμε είναι μη προβλέψιμες. Στην πράξη, υπάρχουν δύο βασικές πηγές κβαντομηχανικής τυχαιότητας: η κβαντομηχανική σε επίπεδο ατόμου και ο θερμικός θόρυβος. Η κβαντομηχανική προβλέπει ότι διάφορα φυσικά φαινόμενα, όπως η πυρηνική διάσπαση του ατόμου, είναι απολύτως τυχαία και δεν μπορούν να προβλεφθούν. Και λόγω πεπερασμένης, μη-μηδενικής θερμοκρασίας κάθε σύστημα χαρακτηρίζεται από τυχαία διακύμανση στην κατάστασή του. Για παράδειγμα, τα μόρια του αέρα κινούνται διαρκώς με τυχαίο τρόπο. Αυτή η τυχαιότητα είναι κβαντικό φαινόμενο.

Εξαιτίας του συμπεράσματος ότι τα κβαντομηχανικά φαινόμενα δεν μπορούν να προβλεφθούν, αποτελούν τις πιο διαδεδομένες γεννήτριες τυχαίων αριθμών. Παραδείγματα τέτοιων κβαντομηχανικών φαινομένων είναι τα παρακάτω:

- Θόρυβος Βολής (shot noise). Ο θόρυβος αυτός δημιουργείται από τις τυχαίες μεταβολές στις αφίξεις των ηλεκτρονίων (ή οπών) και με τον τρόπο αυτό εμφανίζεται στην έξοδο ένας τυχαία μεταβαλλόμενος θόρυβος ρεύματος. Απλό παράδειγμα είναι το φως μιας λάμπας σε μια φωτοδίοδο. Η συλλογή του θορύβου είναι δύσκολο θέμα, αλλά γενικά πρόκειται για μια αρκετά απλή πηγή θορύβου.
- Πηγή ακτινοβολίας από πυρηνική διάσπαση, μετρούμενη με τη βοήθεια ενός μετρητή Geiger συνδεδεμένο σε Η/Υ (βλέπε Fourmilab Hotbits Project [20]).

[1] Πηγάζει από ηλεκτρομαγνητικά κύματα που μεταδίδονται από καλώδια που είναι πολύ κοντά το ένα με το άλλο ή από αθωράκιστα καλώδια.

- Φωτόνια που ταξιδεύουν μέσα από ημι-διάφανο καθρέφτη (Σχήμα 4.1), όπως στο εμπορικό προϊόν Quantis (εταιρεία ID Quantique, Παν/μιο Γενέβης). Τα αμοιβαίως αποκλειώμενα φαινόμενα ανάκλασης και διείσδυσης εντοπίζονται και αντιστοιχίζονται σε "0" ή "1" δυαδικές τιμές.



Σχήμα 4.1 Quantis - Quantum Random Number Generator.

Τα θερμικά φαινόμενα ανιχνεύονται πιο εύκολα, αλλά είναι ευπαθή σε επιθέσεις, ρίχνοντας τη θερμοκρασία του συστήματος σε επίπεδα όπου σταματούν να λειτουργούν με συνέπεια τη μείωση της τυχαιότητάς τους. Οι πιο συνηθισμένες μέθοδοι παραγωγής είναι:

- Θερμικός θόρυβος αντίστασης, ενισχυμένος ώστε να παράγει τυχαία τάση.
- Θόρυβος χιονοστοιβάδας παραγόμενος από μια δίοδο χιονοστιβάδας, ή Zener breakdown noise από μια ανάστροφα πολωμένη δίοδο Zener (US Patent 4853884).
- Ατμοσφαιρικός θόρυβος, που ανιχνεύεται από ραδιοδέκτη συνδεδεμένο με έναν υπολογιστή (random.org).

Ένα άλλο φυσικό φαινόμενο, το οποίο είναι εύκολο να μετρηθεί είναι ο *ρυθμός απόκλισης* (drift) του ρολογιού. Η τυχαιότητα εξαρτάται απόλυτα από τα εξαρτήματα που χρησιμοποιούνται, το σχεδιασμό, την παλαιότητα και τις ρυθμίσεις της συσκευής. Πολλά από αυτά όμως, μπορεί να προβλεφθούν με αποτέλεσμα οι ακολουθίες που παράγονται να είναι περισσότερο προβλέψιμες.

4.3.1 Lavarnd.org

Άλλοι έχουν προτείνει χρήση ψηφιακής κάμερας για την εκμετάλλευση χαοτικών μακροσκοπικών φαινομένων κάποιας φωτογραφίας. Η δυσκολία έγκειται στον εντοπισμό αυτών των σχημάτων χαοτικής φύσης, που δημιουργούν πραγματικά τυχαίους αριθμούς. Μια ομάδα της Silicon Graphics απεικόνισε λάμπες φθορίου για τη δημιουργία τυχαίων αριθμών (U.S. Patent 5732138). Η

ψηφιοποιημένη εικόνα που περιέχει δομημένα δεδομένα και χαοτικό θόρυβο αποτελεί είσοδο στον αλγόριθμο Digital Blender. Με την εφαρμογή *n* διαφορετικών SHA-1 συναρτήσεων κατακερματισμού που τρέχουν παράλληλα και *n* διαφορετικών πράξεων xor περιστροφής και αναδίπλωσης (xor rotate and fold) σε δεδομένα που περιέχουν χαοτικό θόρυβο, καταστρέφουν το δομημένο τμήμα και παράγουν ομοιόμορφα κατανεμημένα τυχαία δεδομένα.

4.3.2 Intel 80802 Firmware Hub chip

Το 80802 Firmware Hub της Intel (μέρος των chipset 800,840) περιλαμβάνει μια hardware RNG με χρήση ενός αργού και ενός γρήγορου ταλαντωτή. Μια θερμική πηγή θορύβου χρησιμοποιείται για να διαμορφώσει τη συχνότητα του αργού ταλαντωτή, ο οποίος στη συνέχεια, προκαλεί μέτρηση του γρήγορου ταλαντωτή. Από την έξοδο αυτή, αφαιρείται η πόλωση με χρήση του αλγόριθμου αποσυσχέτισης Von Neumann (βλέπε §4.4). Η συσκευή έχει ρυθμό εξόδου περίπου 100kbps.

4.3.3 Μικροεπεξεργαστές VIA C3

Από το 2003, όλοι οι VIA C3 μικροεπεξεργαστές έχουν μια hardware RNG (VIA PadLock Security Engine). Αντί της χρήσης θερμικού θορύβου, χρησιμοποιούνται και εδώ ταλαντωτές που λειτουργούν σε διαφορετικό ρυθμό. Η έξοδοι των δύο, γίνονται XOR για να ελέγξουν την πόλωση ενός τρίτου ταλαντωτή, του οποίου η έξοδος ενεργοποιεί έναν τέταρτο για την παραγωγή των bits. Μικρές μεταβολές στη θερμοκρασία, και σε άλλες τοπικές συνθήκες προκαλούν τυχαίες διακυμάνσεις στους ταλαντωτές παράγοντας έτσι τυχαίες ακολουθίες από bit. Για τη διασφάλιση της τυχαιότητας υπάρχουν δύο τέτοιες RNG σε κάθε επεξεργαστή και η τελική έξοδος λαμβάνεται από τη μίξη αυτών των δύο γεννητριών. Ο ρυθμός εξόδου είναι 10-100Mbps.

4.3.4 CryptoLib

Μια υλοποίηση της παραπάνω ιδέας σε λογισμικό, περιλαμβάνεται στη βιβλιοθήκη κρυπτογράφησης CryptoLib [21]. Ο αλγόριθμος βασίζεται στη συνάρτηση truerand() του Unix. Οι περισσότεροι σύγχρονοι υπολογιστές έχουν δύο ταλαντωτές κρυστάλλου, έναν για το ρολόι πραγματικού χρόνου (για τα timer interrupts) και έναν για το πρωτεύων ρολόι της CPU. Η truerand() χρησιμοποιεί ένα service του ΛΣ το οποίο ελέγχει ένα alarm. Μια υπορουτίνα θέτει το alarm σε off (π.χ. κάθε 1/60 sec). Στη συνέχεια, μια άλλη μπαίνει σε ένα βρόχο while σε αναμονή της ενεργοποίησης του alarm. Καθώς το alarm ενεργοποιείται κατά τυχαίο τρόπο (και όχι σε κάθε χτύπο),

το ελάχιστο σημαντικότερο ψηφίο του μετρητή των επαναλήψεων του βρόχου, για το διάστημα αυτό, μεταβάλλεται τυχαία. Η truerand() δεν απαιτεί επιπλέον hardware.

4.3.5 Random.org

Η TRNG του Random.org [19], η οποία χρησιμοποιήθηκε και στην παρούσα εργασία, σχεδιάστηκε από το Τμήμα Επιστήμης Υπολογιστών του Παν/μίου Trinity και κάνει χρήση του ατμοσφαιρικού θορύβου. Ο θόρυβος αυτός προέρχεται από το συντονισμό ενός ραδιοφώνου σε συχνότητα που δεν χρησιμοποιείται. Στη συνέχεια, ενισχύεται και αναπαράγεται σε υπολογιστή όπου με χρήση κατάλληλου λογισμικού μετατρέπεται σε 8-bit mono σήμα με συχνότητα 8kHz. Τα πρώτα επτά ψηφία απομακρύνονται και τα εναπομείναντα συγκεντρώνονται. Πριν η ακολουθία λάβει την τελική της μορφή, εκτελείται skew correction με χρήση του αλγορίθμου John Von Neumann (βλέπε §4.4) για να διασφαλιστεί ομοιόμορφη κατανομή των ψηφίων.

4.3.6 Τυχαιότητα με χρήση Υπολογιστή

Εκτός από τα κβαντικά φαινόμενα και το θερμικό θόρυβο, μπορούν να χρησιμοποιηθούν και άλλα φαινόμενα τα οποία τείνουν προς την τυχαιότητα, παρότι δεν χαρακτηρίζονται εύκολα από νόμους της φύσης. Όταν πολλές τέτοιες πηγές συνδυάζονται κατάλληλα, μπορεί να συλλεγεί αρκετή εντροπία για τη δημιουργία κρυπτογραφικών κλειδιών (βλέπε αλγόριθμο Yarrow-160 ή Fortuna [22]). Το πλεονέκτημα αυτής της προσέγγισης είναι η έλλειψη αναγκαιότητας ειδικού hardware. Ιδιαίτερη προσοχή όμως πρέπει να δωθεί στην on-line παραγωγή τυχαίων αριθμών καθώς οι παραπάνω τιμές είναι ευπαθείς σε επιθέσεις spoofing.

Τυπικά, σε τέτοιου είδους γεννήτριες χρησιμοποιούνται πηγές από interrupts που προκαλούν διάφορες συσκευές. Κλασσικό παράδειγμα, τυχαίας ακολουθίας είναι τα λιγότερο σημαντικά ψηφία του χρόνου μεταξύ των keystrokes ενός χρήστη. Παρόμοιες προσεγγίσεις κάνουν χρήση μετρήσεων όπως task-scheduling, network hits, disk-head seek times κ.α. Η συνάρτηση Microsoft's® CryptGenRandom (μέρος του CryptoAPI), περιλαμβάνει έναν μακρύ κατάλογο από τιμές που θα μπορούσαν να χρησιμοποιηθούν όπως process ID, thread ID, τρέχουσα τοπική ώρα, MD4 hash τιμών περιβάλλοντος χρήστη όπως username, computer name και search path, μετρητές CPU (RDTSC, RDMSR, RDPMC), κ.α. Παρόμοιο σχεδιασμό έχει και η βιβλιοθήκη "/dev/random" του Linux.

4.4 Απαλειφη Πολωσης

Οι ακολουθίες ψηφίων των συστημάτων παραγωγής τυχαίων αριθμών είναι επιρρεπείς σε πολώσεις (κυριαρχία 1 ή 0). Υπάρχουν δύο προσεγγίσεις για την εξάλειψη της πόλωσης. Η μία κάνει χρήση του Θεωρήματος Κεντρικού Ορίου (Central Limit Theorem), βάσει του οποίου η ανατροφοδότηση του σήματος εξόδου, το οποίο φιλτράρεται από χαμηλοπερατό φίλτρο, εξαλείφει (αλλά όχι πλήρως) το πολωμένο σήμα. Η μέθοδος αυτή χρησιμοποιείται συχνά από γεννήτριες υψηλών ταχυτήτων καθώς η διόρθωση του σήματος γίνεται on-line.

Η δεύτερη προσέγγιση γίνεται off-line μετά τη δημιουργία της ακολουθίας. Υπάρχουν διάφορες τεχνικές, γνωστές ως αλγόριθμοι «whitening»:

Ο John V. Neumann, επινόησε έναν απλό αλγόριθμο για τη διόρθωση της πόλωσης και τη μείωση της συσχέτισης μεταξύ των αριθμών. Χωρίζει την ακολουθία σε ζεύγη bit. Όταν υπάρχει μετάβαση, π.χ. 01 ή 10, το δεύτερο bit αφαιρείται και η ακολουθία γίνεται 0 ή 1 αντίστοιχα. Στην περίπτωση που δύο διαδοχικά bits είναι ίδια, αφαιρούνται και τα δύο από την ακολουθία. Η μέθοδος αυτή προσεγγίζει ικανοποιητικά τη συχνότητα εμφάνισης των 0 και 1 στο 50% και είναι εύκολα υλοποιήσιμη.

Μια άλλη τεχνική βελτίωσης είναι να γίνει XOR η ακολουθία με την έξοδο μιας δεύτερης ψευδοτυχαίας γεννήτριας (π.χ. Blum Blum Shub). Αυτό μπορεί να βελτιώσει, με μικρό κόστος, την αποσυσχέτιση και την πόλωση. Παρόμοια μέθοδος είναι να γίνουν XOR δύο ή περισσότερες ασυσχέτιστες ακολουθίες. Αν η πιθανότητα εμφάνισης του 0 είναι $1/2 + e$, όπου e η πόλωση της ακολουθίας και $-1/2 \leq e \leq 1/2$, η πόλωση του αποτελέσματος XOR θα είναι $2e^2$ (βλέπε λήμμα Piling-up).

Άλλες προσεγγίσεις εφαρμόζουν συναρτήσεις κατακερματισμού όπως MD5, SHA-1, ή RIPEMD-160 ή ακόμα και CRC σε όλη ή σε μέρος της ακολουθίας. Τέτοιες μέθοδοι αυτή είναι ιδιαίτερα ελκυστικές λόγω ταχύτητας, αλλά εξαρτώνται εξ ολοκλήρου από την έξοδο της συνάρτησης για την οποία υπάρχει μικρή θεωρητική βάση.

Η σελίδα αυτή είναι σκόπιμα λευκή.

5. ΠΡΩΤΟΚΟΛΛΟ SECUREMEM/OTP

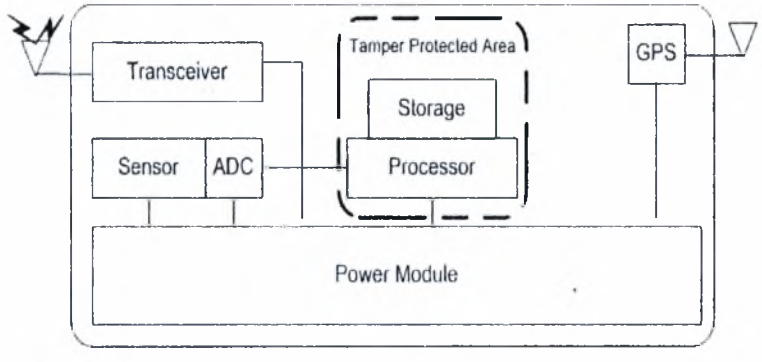
5.1 ΕΙΣΑΓΩΓΗ

Στην παρούσα εργασία, υλοποιήθηκε το δεύτερο μέλος της οικογένειας πρωτοκόλλων *SecureMem* (βλέπε Κεφ. 2), η οποία αποτελείται από δύο μέλη, το *SecureMem* και το *SecureMem/OTP* [23].

Τα πρωτόκολλο *SecureMem/OTP*, εξασφαλίζει την ασφαλή διακίνηση δεδομένων στα δίκτυα αισθητήρων, βασιζόμενο στη χρήση 1) ενός ισχυρού αλγορίθμου για την κρυπτογράφηση των δεδομένων, 2) μεθόδου συμμετρικής κρυπτογραφίας για τη μετάδοση των κρυπτογραφημάτων, και 3) κόμβων ανθεκτικών σε φυσική παραβίαση. Στις επόμενες παραγράφους θα περιγράψουμε αναλυτικά το πρωτόκολλο, *SecureMem/OTP*.

5.2 ΕΠΙΘΥΜΗΤΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΚΟΜΒΩΝ

Δύο είναι τα επιθυμητά χαρακτηριστικά, επιπλέον εκείνων που αναφέραμε στις προηγούμενες ενότητες, τα οποία πρέπει να διαθέτουν οι κόμβοι (Σχήμα 5.1), ώστε να μπορεί να υλοποιηθεί σε αυτούς το πρωτόκολλο *SecureMem/OTP*.



Σχήμα 5.1 Υποσυστήματα Κόμβου με Αυξημένη Αντοχή Ενάντια στη Φυσική Παραβίαση.

Το πρώτο και βασικότερο είναι η αντοχή σε τυχόν προσπάθειες φυσικής παραβίασης τους. Για να επιτευχθεί ο παραπάνω στόχος, πρέπει να υποστηρίζονται από τον κόμβο κάποια στοιχεία τα οποία μπορούν να είναι είτε ενεργητικά είτε παθητικά, και των οποίων οι βασικοί στόχοι είναι: 1) η

προστασία του υποσυστήματος αποθήκευσης, εντός του οποίου αποθηκεύεται το κρυπτογραφικό υλικό, 2) η απαγόρευση ανάλυσης της λειτουργίας του κόμβου από απόσταση, και 3) σε περίπτωση που ο κόμβος περιέλθει στην κατοχή κάποιου κακόβουλου χρήστη, η αυτοκαταστροφή του ώστε να μη δοθεί η ευκαιρία εκμετάλλευσης του κόμβου με οποιοδήποτε τρόπο.

Όπως είπαμε τα συστήματα αυτοπροστασίας του κόμβου μπορούν να είναι ενεργητικά ή παθητικά. Στα παθητικά συμπεριλαμβάνονται αδιάρρηκτες ειδικές συσκευασίες από ενισχυμένο ατσάλι οι οποίες ασφαλίζονται με ειδικές βίδες μονής κατεύθυνσης. Στα ενεργητικά, περιλαμβάνονται ειδικές διατάξεις οι οποίες καταστρέφουν τον κόμβο όταν γίνει παραβίαση της εξωτερικής συσκευασίας του. Η καταστροφή του κόμβου μπορεί να γίνει με διάφορους τρόπους, π.χ. με μικρή ποσότητα εκρηκτικών (σε κόμβους που χρησιμοποιούνται σε στρατιωτικές εφαρμογές), με την διάλυση μικροκάψουλων με ισχυρές διαλυτικές χημικές ουσίες στο εσωτερικό τους, ή με την παροχή υψηλής έντασης ρεύματος ή τάσης τροφοδοσίας στα κυκλώματα του.

Το δεύτερο επιθυμητό χαρακτηριστικό, είναι η ύπαρξη ενός υποσυστήματος αποθήκευσης δεδομένων στον κόμβο. Αυτό πρέπει να απαιτεί για τη λειτουργία του, χαμηλή κατανάλωση ενέργειας και ταυτόχρονα να έχει τέτοιο μέγεθος ώστε να μπορούν να προ-αποθηκευθούν σε αυτό όσα κλειδιά χρειάζονται για τη λειτουργία του κόμβου, για το συνολικό χρονικό διάστημα της επιχειρησιακής του ζωής. Και οι δύο αυτές απαιτήσεις μπορούν να ικανοποιηθούν από τις μνήμες που υπάρχουν διαθέσιμες στην αγορά για χρήση στα δίκτυα αισθητήρων.

Πίνακας 5.1 Είδη και Αποθηκευτική Ικανότητα Διάφορων Αποθηκευτικών Μέσων.

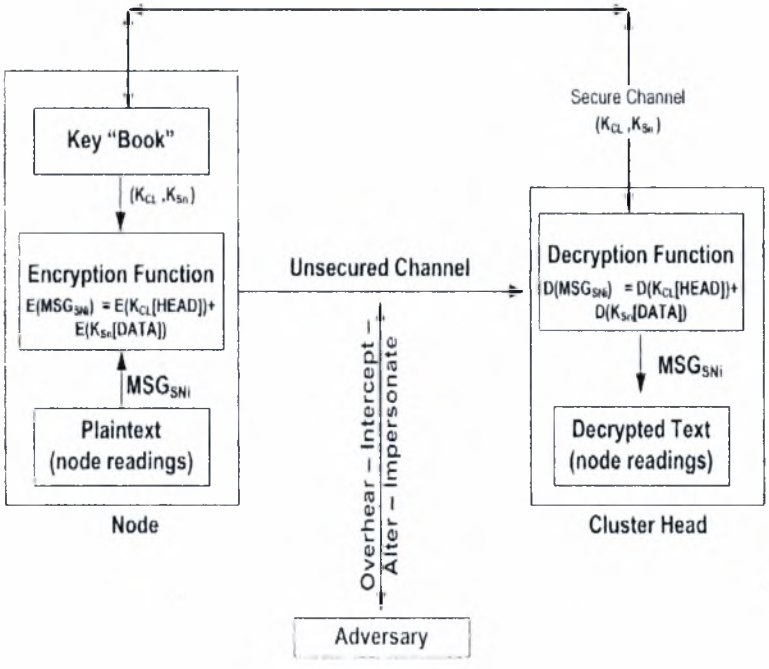
Είδος Αποθηκευτικού Μέσου	**Αποθηκευτική Ικανότητα**	**Ικανότητα Αποθήκευσης Αρχείων Word**
Removable USB Drive	64MB	600, 50-page documents
Removable USB Drive	128MB	1.250, 50-page documents
Removable USB Drive	1GB	10.000, 50-page documents
CD	650MB	6.500, 50-page documents
DVD	4.6GB	46.000, 50-page documents

Στον Πίνακα 5.1 βλέπουμε τη χωρητικότητα ορισμένων τυπικών αποθηκευτικών μέσων. Το σημείο στο οποίο πρέπει να εστιάσουμε, είναι η αποθηκευτική ικανότητα των USB τύπου FLASH δίσκων. Εκδοχές των δίσκων αυτών με χαρακτηριστικά χαμηλής κατανάλωσης ενέργειας, υπάρχουν

διαθέσιμες και μπορούν να χρησιμοποιηθούν στους κόμβους των δικτύων αισθητήρων, παρέχοντας χωρητικότητα ικανή να υποστηρίξει τη λειτουργία τους για όλο τον κύκλο της ζωής τους.

5.3 Μεθοδος Κρυπτογραφησης

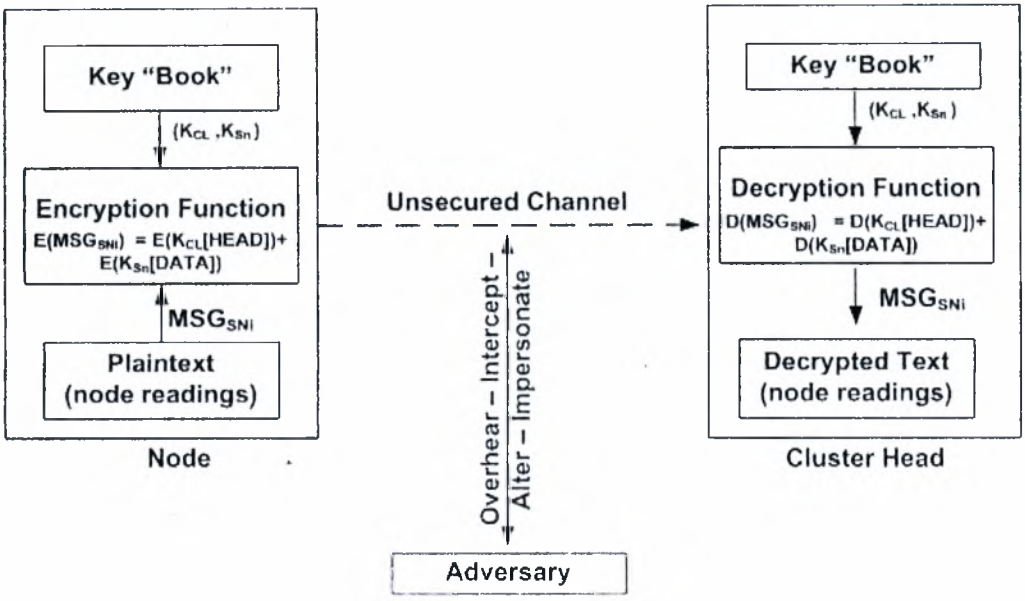
Όπως είπαμε, πολλά πρωτόκολλα στηρίζονται, για την ασφαλή μεταφορά των δεδομένων, στην κρυπτογραφία συμμετρικού κλειδιού (Σχήμα 5.2). Αυτό σημαίνει ότι για την κρυπτογράφηση και αποκρυπτογράφηση των δεδομένων που διακινούνται στο δίκτυο, χρησιμοποιείται ένα κλειδί το οποίο είναι γνωστό τόσο στον αποστολέα όσο και στον παραλήπτη. Στη γενική περίπτωση, η κρυπτογραφημένη επικοινωνία με τη χρήση μυστικού κλειδιού απαιτεί την ύπαρξη ενός ασφαλούς καναλιού, διαμέσω του οποίου ο αποστολέας κοινοποιεί το μυστικό κλειδί στον παραλήπτη.



Σχήμα 5.2 Επικοινωνία με Χρήση Μυστικού Κλειδιού.

Όμως, τις περισσότερες φορές δεν είναι δυνατή η εξασφάλιση ενός τέτοιου καναλιού, ενώ εάν αυτό υφίσταται, τα δύο μέλη τα οποία συμμετέχουν στην ανταλλαγή του μυστικού κλειδιού δεν μπορούν να είναι απολύτως σίγουρα για τη «στεγανότητά» του.

Το πρωτόκολλο SecureMem/OTP, παρακάμπτει αυτό το πρόβλημα με τον προϋπολογισμό και την αποθήκευση όλων των κλειδιών, σε προστατευόμενες ασφαλείς περιοχές αποθήκευσης μέσα σε κάθε κόμβο (Σχήμα 5.3).



Σχήμα 5.3 Επικοινωνία με Χρήση του Πρωτοκόλλου SecureMem/OTP.

5.3.1 Επισκόπηση Πρωτοκόλλου

Το πρωτόκολλο αποτελείται από τρεις φάσεις κατά τη διάρκεια των οποίων συγκεκριμένες ενέργειες πρέπει να λάβουν χώρα. Οι φάσεις αυτές διακρίνονται στις εξής: 1) πριν την ανάπτυξη του δικτύου, 2) κατά την αρχικοποίηση του, και 3) κατά την κανονική του λειτουργία.

Κατά την πρώτη φάση, δημιουργείται ένα «βιβλίο» από τυχαία κλειδιά στο σταθμό βάσης. Το «βιβλίο» αυτό είναι ακριβώς το ίδιο και αποθηκεύεται στο υποσύστημα αποθήκευσης κάθε κόμβου, έχοντας ως σκοπό να λειτουργεί σαν θησαυροφυλάκιο από όπου οι κόμβοι θα διαλέγουν τα κρυπτογραφικά κλειδιά τους κατά τη διάρκεια της επιχειρησιακής τους δράσης. Επιπλέον, ο σταθμός βάσης δημιουργεί ένα κλειδί κοινό για όλο το δίκτυο, το οποίο θα χρησιμοποιηθεί από τους κόμβους κατά τη φάση αρχικοποίησης του δικτύου, κατά τη διάρκεια της οποίας δημιουργείται το δίκτυο κορμού (Σταθμός Βάσης - Επικεφαλής Ομάδων, Επικεφαλής Ομάδων - Κόμβοι). Μόλις ολοκληρωθεί η δημιουργία του δικτύου κορμού και τεθεί σε κανονική λειτουργία, κάθε μήνυμα που μεταδίδεται πρός και από το σταθμό βάσης κρυπτογραφείται εφαρμόζοντας τον αλγόριθμο OTP που είδαμε σε προηγούμενο κεφάλαιο.

5.4 Αναλυτικη Παρουσιαση Πρωτοκολλου

Πριν συνεχίσουμε με την αναλυτική παρουσίαση του πρωτοκόλλου, στον Πίνακας 5.2 παρουσιάζονται οι βασικοί συμβολισμοί που χρησιμοποιούνται σε αυτά.

Συμβολισμος	Ερμηνεια
BS	Σταθμός Βάσης
Ch	Επικεφαλής Κόμβος
Sn	Απλός Κόμβος
[Head]	Επικεφαλίδα Μηνύματος
[Data]	Δεδομένα μηνύματος
MSG_i	Μήνυμα όπου i= {BS, Ch , Sn}
Id	Μοναδικός αριθμός αναγνώρισης ταυτότητας κόμβου
E	Συνάρτηση Κρυπτογράφησης
$K_{NL(\mu)}$	Κλειδί Επιπέδου Δικτύου [μ = { (n)ew, (o)ld }]
$K_{Ch(\mu)}$	Κλειδί του Επικαφαλής Ομάδας [μ = { (n)ew, (o)ld }]
$K_{Sn(\mu)}$	Κλειδί Επιπέδου Κόμβου, where μ = { (n)ew, (o)ld }

5.4.1 Φάση Πριν την Ανάπτυξη (Pre-deployment Phase)

Κατά τη φάση αυτή δημιουργείται ένα «βιβλίο» από τυχαία κλειδιά στο σταθμό βάσης. Το «βιβλίο» αυτό είναι ακριβώς το ίδιο και αποθηκεύεται στο υποσύστημα αποθήκευσης κάθε κόμβου, έχοντας ως σκοπό να λειτουργεί σαν θησαυροφυλάκιο από όπου οι κόμβοι θα διαλέγουν τα κρυπτογραφικά κλειδιά τους κατά τη διάρκεια της επιχειρησιακής τους ζωής. Η παραγωγή των κλειδιών απαιτείται να γίνει από κάποια γεννήτρια πραγματικά τυχαίων αριθμών (βλέπε Κεφάλαιο 4).

Παράλληλα, με το βιβλίο των κλειδιών, σε κάθε κόμβο αποθηκεύεται και το κλειδί επιπέδου δικτύου K_{NL}, το οποίο είναι απαραίτητο για την αυθεντικοποίηση των κόμβων κατά την έναρξη της κανονικής λειτουργίας του δικτύου. Το επιλεχθέν K_{NL} παραμένει σε ισχύ, έως ότου ο χειριστής του δικτύου ή κάποιος εξουσιοδοτημένος κόμβος αποφασίσει να το αλλάξει.

Η δημιουργία των κλειδιών πριν την ανάπτυξη του δικτύου επιτρέπει τη σημαντική εξοικονόμηση ενέργειας για τους κόμβους, μιας και ο υπολογισμός ενός ισχυρού κρυπτογραφικά κλειδιού, απαιτεί την εκτέλεση μια σειράς από επαναλαμβανόμενους σύνθετους υπολογισμούς, οι οποίοι είναι ιδιαίτερα απαιτητικοί σε υπολογιστική ισχύ και επομένως ιδιαίτερα δαπανηροί ως προς την κατανάλωση ενέργειας.

5.4.2 Φάση Αρχικοποίησης (Initialization Phase)

Μετά από την ανάπτυξη του δικτύου στην περιοχή λειτουργίας λαμβάνουν χώρα σε αυτό, οι παρακάτω ενέργειες:

1. Κάθε *Ch* και *Sn* ξεκινά τη λειτουργία του, εκτελεί αυτοέλεγχο των συστημάτων του, και με τη χρήση του *GPS* δέκτη του, υπολογίζει τις ακριβείς γεωγραφικές συντεταγμένες του.
2. Ο *BS* στέλνει ένα μήνυμα στον *Ch* με το οποίο του ζητά να του διαβιβάσει την ταυτότητα και τη θέση όλων των κόμβων *Sn* οι οποίοι σχηματίζουν την ομάδα του. Η σχέση η οποία περιγράφει αυτές τις ενέργειες είναι η ακόλουθη:

$$MSG_{BS} = [Head] \quad (5.1)$$

Όπως είναι εμφανές, ο *BS* στέλνει μόνο μια επικεφαλίδα, στην οποία περιέχεται μια ένδειξη για τη διάκριση του τύπου μηνύματος σε probe message. Το πεδίο *SensorID* στην επικεφαλίδα είναι μοναδικό για κάθε κόμβο και κάθε μήνυμα για να επεξεργαστεί από τον *Ch* και τον *BS* πρέπει να έχει προέλευση από έναν *Sn* ο οποίος είναι καταγεγραμμένος στο αρχείο κόμβων του *Ch* και του *BS* αντίστοιχα· Οι κόμβοι προστίθενται σε αυτό κατά τη φάση της αρχικοποίησης, με μια διαδικασία που θα δούμε στη συνέχεια. Η προσέγγιση αυτή εξασφαλίζει την αυθεντικοποίηση των κόμβων.

Πίνακας 5.3 Πεδία Επικεφαλίδας και Περιγραφή.

Όνομα	Ερμηνεία	Τιμές τις οποίες λαμβάνει	Μέγεθος σε (bytes)
Χρονοσφραγίδα[2]	Ακριβής ώρα δημιουργίας του μηνύματος	hh/mm/ss	3
	Ημέρα δημιουργίας του μηνύματος	dd/mm/yy	3
Τελικός Αποδέκτης	Μοναδικό ID παραλήπτη	0x07QW ή 0xFFFF (broadcast)	2
Τύπος Μηνύματος	Διακριτικό μηνύματος για το είδος της εντολής από τον BS	01	1
Αποστολέας (SensorID)	Αναγνωριστικό κόμβου	0x8AC2	2
Γεωγραφικό Μήκος Κόμβου (x)[2]	Γεωγραφικό Μήκος Κόμβου (E/W) standard GPS format	234256E	4

[2] Τα πεδία αυτά είναι προαιρετικά ανάλογα με το είδος της εφαρμογής. Η παρούσα υλοποίηση δεν συμπεριλαμβάνει τα συγκεκριμένα πεδία.

Γεωγραφικό Πλάτος Κόμβου (y)[2]	Γεωγραφικό Πλάτος Κόμβου (N/S) standard GPS format	335645N	4
Ύψος Κόμβου (z)[2]	Ύψος σε μέτρα, από την επιφάνεια της θάλασσας.	0x4B00	2
Διεύθυνση Κλειδιού στην Pad	Διεύθυνση από την οποία πρέπει κάποιος να αρχίσει να διαβάζει για να αποκρυπτογραφήσει το μήνυμα	0x6DC0	2
Κλειδί K_{NL}	Κλειδί επιπέδου δικτύου για την αυθεντικοποίηση των κόμβων	0x9F63	2

3. Κάθε *Ch* ο οποίος λαμβάνει το παραπάνω μήνυμα από τον *BS*, στέλνει ένα μήνυμα με το οποίο ζητά από οποιονδήποτε κόμβο το λάβει να αναφέρει σε αυτόν, τα στοιχεία ταυτότητάς του και τη θέση του.
4. Κάθε *Sn* ο οποίος λαμβάνει το παραπάνω μήνυμα, απαντά στέλνοντας νέο μήνυμα στον *Ch* κρυπτογραφημένο αυτή τη φορά, με όλες τις πληροφορίες που του ζητήθηκαν (ταυτότητα, γεωγραφικές συντεταγμένες) και το κλειδί K_{NL}.

$$MSG_{Sn} = E(K_{Sn}, [Head]) \tag{5.2}$$

5. Όταν ο *Ch* λάβει όλα τα μηνύματα από τους *Sn* οι οποίοι ανήκουν στην ομάδα του (την οποία γνωρίζει εκ των προταίρων) τότε το αποκρυπτογραφεί και:
   - ☐ Αποθηκεύει, την ταυτότητα κάθε κόμβου μαζί με τις γεωγραφικές συντεταγμένες του (x,y,z) σε ένα νέο αρχείο-ευρετήριο της ομάδας του, το οποίο αντιπροσωπεύει τους ενεργούς αυθεντικοποιημένους κόμβους της ομάδας.
   - ☐ Δημιουργεί κρυπτογραφημένο μήνυμα με το οποίο γνωστοποιεί στον *BS* τα περιεχόμενα του αρχείου-ευρετηρίου των κόμβων που ανήκουν στην ομάδα του, συμπεριλαμβάνοντας στο μήνυμα και το K_{NL} στο πεδίο Data για την αυθεντικοποίησή του από τον *BS*. Η γενική συνάρτηση η οποία περιγράφει αυτές τις ενέργειες είναι η ακόλουθη. Σημειώνεται ότι οι δείκτες i και j δηλώνουν τη θέση του κλειδιού στην pad.

$$MSG_{Ch} = E\left(K_{Ch}^{i}, [Head]\right) + E\left(K_{Ch}^{j}, [Data]\right) \tag{5.3}$$

6. Όταν ο *BS* λάβει όλα τα μηνύματα από τους *Ch*, τα αποκρυπτογραφεί, και αφού γίνει η αυθεντικοποίηση των *Ch*, αποθηκεύει την ταυτότητα κάθε *Sn* και *Ch* μαζί με τις γεωγραφικές συντεταγμένες τους (x,y,z) σε ένα αρχείο-ευρετήριο στο οποίο περιλαμβάνεται όλο το δίκτυο.
7. Δημιουργούνται τα αρχεία-ευρετήρια κόμβων σε επίπεδο ομάδας και δικτύου.
8. Όλοι οι κόμβοι εκκινούν την κανονική τους λειτουργία.

5.4.3 Φάση Κανονικής Λειτουργίας (Regular Operation)

Κατά την κανονική λειτουργία του δικτύου μπορούν να συμβούν τα παρακάτω:

1. ***Προσθήκη ενός νέου κόμβου σε μια ομάδα:*** όταν ένας *Ch* δεχτεί ένα αίτημα από έναν *Sn*, για την ένταξή του στην ομάδα που αυτός είναι επικεφαλής. Για να γίνει αυτό αποδεκτό πρέπει να συντρέχουν οι παρακάτω συνθήκες: α) το μήνυμα να περιέχει κρυπτογραφημένο το ισχύον τη στιγμή του αιτήματος κλειδί K_{NL} και β) το μοναδικό αναγνωριστικό του κόμβου *SensorID* πρέπει να υπάρχει καταχωρημένο στο ευρετήριο κόμβων του δικτύου. Εάν ικανοποιούνται και οι δύο συνθήκες, ο κόμβος θα γίνει αποδεκτός από τον επικεφαλής ομάδας, και τα στοιχεία ταυτότητάς του θα προστεθούν στο ευρετήριο κόμβων.
2. ***Συλλογή και μετάδοση δεδομένων:*** *από έναν κόμβο προς τον επικεφαλής της ομάδας στην οποία ανήκει:* Η δομή του μηνύματος το οποίο στέλνει ο *Sn* στον *Ch* και εν συνεχεία ο *Ch* στον *BS*, φαίνεται στις Σχέσεις (5.4) και (5.5). Στο πεδίο Head συμπεριλαμβάνεται το κλειδί K_{NL} και η διεύθυνση του κλειδιού στην Pad από την οποία ο παραλήπτης πρέπει να αρχίσει να αποκρυπτογραφεί, ενώ το πεδίο Data περιλαμβάνει τις τιμές που συλλέγει ο αισθητήρας.

$$MSG_{Sn} = E\left(K_{Sn}^{i}, [Head]\right) + E\left(K_{Sn}^{j}, [Data]\right) \quad (5.4)$$

$$MSG_{Ch} = E\left(K_{Ch}^{i}, [Head]\right) + E\left(K_{Ch}^{j}, [Data]\right) \quad (5.5)$$

3. ***Αλλαγή των κλειδιών κρυπτογράφησης:*** Η αλλαγή του K_{NL}, και των κλειδιών της Pad κάθε κόμβου στα πλαίσια του *SecureMem/OTP,* μπορεί να προκληθεί είτε με την επέμβαση του χειριστή του *BS,* είτε ακολουθώντας ένα συγκεκριμένο χρονοδιάγραμμα το οποίο έχει καταρτιστεί πριν από την ανάπτυξη του δικτύου, με βάση τα χαρακτηριστικά της εφαρμογής στα πλαίσια της οποίας το δίκτυο επιχειρεί.
4. ***Διαγραφή «ύποπτου» κόμβου:*** Αν και εξαιτίας της ειδικής κατασκευής των κόμβων, ώστε να είναι ανθεκτικοί σε κάθε προσπάθεια φυσικής παραβίασής τους, θεωρείται απίθανο να καταφέρει ένας κακόβουλος χρήστης να θέσει κάποιον από τους κόμβους υπό τον έλεγχο του. Στην περίπτωση όμως που υπάρξει υποψία για κάτι τέτοιο, ο *BS* διαγράφει τον ύποπτο κόμβο από το ευρετήριο των κόμβων του δικτύου και δημιουργεί ένα νέο $K_{NL(n)}$ το οποίο στη συνέχεια το γνωστοποιεί στους κόμβους της ομάδας του, με ειδικού τύπου μήνυμα το οποίο λέγεται "revocation message". Με αυτό το μήνυμα, οι *Ch* διαγράφουν με τη σειρά τους τον ύποπτο κόμβο

από το ευρετήριό τους και προωθούν το νέο $K_{NL(n)}$ στους κόμβους της ομάδας τους. Αυτό σημαίνει ότι ο επικεφαλής μιας ομάδας, η οποία αποτελείται από *n* μέλη, θα στείλει συνολικά *n-1* μηνύματα.

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΜΕΡΟΣ ΙΙ - ΕΦΑΡΜΟΓΗ

Η σελίδα αυτή είναι σκόπιμα λευκή.

6. ΥΛΟΠΟΙΗΣΗ

6.1 ΕΡΓΑΛΕΙΑ ΠΡΟΣΟΜΟΙΩΣΗΣ

6.1.1 TinyOS και η Γλώσσα Προγραμματισμού NesC

Η επιτυχία και η δημοτικότητα που γνωρίζουν τα ασύρματα δίκτυα αισθητήρων τα τελευταία χρόνια οφείλεται σε μεγάλο βαθμό και στο TinyOS [24], ένα λειτουργικό σύστημα που σχεδιάστηκε για δίκτυα τέτοιου τύπου. Βασίζεται στο μοντέλο του component-based προγραμματισμού, καθώς είναι γραμμένο σε γλώσσα nesC [25], μια διάλεκτο της C, βελτιστοποιημένη για τις απαιτήσεις μνήμης των δικτύων αισθητήρων. Τα συμπληρωματικά του εργαλεία είναι γραμμένα κυρίως σε Java και shell scripts, ενώ δεν λείπει και κώδικας γραμμένος σε C (NesC compiler, AVR binutils).

Ένα πρόγραμμα TinyOS αποτελείται από ένα γράφο από components (Σχήμα 6.4), καθένα από τα οποία είναι μια ανεξάρτητη υπολογιστική οντότητα. Η χρήση του μοντέλου εξαρτημάτων (component model) επιτρέπει να συνδυάζονται εύκολα επιμέρους εξαρτήματα παράγωντας ολοκληρωμένες εφαρμογές, στοχεύοντας με αυτόν τον τρόπο σε μεγαλύτερη ευελιξία και προσαρμοστικότητα.

Τα components περιέχουν τρία είδη συναρτήσεων· εντολές (commands), γεγονότα (events), και εργασίες (tasks). Οι εντολές και τα γεγονότα είναι μηχανισμοί επικοινωνίας μεταξύ των components, ενώ τα tasks χρησιμοποιούνται για το συγχρονισμό κώδικα που εκτελείται στο εσωτερικό των components.

Η διασύνδεση των components δείχνει την επικοινωνία μεταξύ τους και τη ροή των events. Αυτή η διασύνδεση ονομάζεται wiring specification και είναι ανεξάρτητη από τα components. Οι συνδέσεις αυτές μεταξύ components, οι οποίες ονομάζονται interfaces, είναι διπλής κατεύθυνσης.

Μια εντολή είναι μια αίτηση προς ένα component να εκτελέσει κάποια λειτουργία, όπως η έναρξη λειτουργίας ενός αισθητήρα. Το event δηλώνει (με signal) την ολοκλήρωση αυτής της λειτουργίας. Τα events μπορούν, επίσης, να προκαλούνται με ασύγχρονο τρόπο, όπως για παράδειγμα, εξαιτίας κάποιου hardware interrupt (π.χ. timer) ή της άφιξης κάποιου μηνύματος.

Οι εντολές και τα events εκτός από το να εκτελούν σύντομες λειτουργίες, μπορούν να εκτελέσουν ένα task, μια ρουτίνα δηλ. που εκτελείται από τον scheduler του TinyOS αργότερα. Ο διαχωρισμός αυτός καθιστά τις εντολές και τα events υπεύθυνα, για τη διεκπεραίωση άμεσων λειτουργιών και τα tasks για πιο εκτεταμένους υπολογισμούς, καθώς τα τελευταία εκτελούνται από το scheduler (FIFO στοίβα), χωρίς προσπεράσεις από άλλα tasks (μπορεί όμως να διακοπεί η εκτέλεση ενός task και να αρχίσει η εκτέλεση ενός event).

6.1.2 Avrora

Για τις μετρήσεις κατανάλωσης ενέργειας χρησιμοποιήθηκε το λογισμικό ανοιχτού κώδικα Avrora [26]. Είναι κατασκευασμένο από το πανεπιστήμιο UCLA και παρέχει τη δυνατότητα προσομοίωσης και ανάλυσης προγραμμάτων, γραμμένων για AVR μικροεπεξεργαστές της Atmel και αισθητήρες Mica2.

Τα κριτήρια επιλογής του συγκεκριμένου εργαλείου είναι η ακρίβεια των αποτελεσμάτων σε συνδυασμό με την επεκτασιμότητα που το χαρακτηρίζει. Αυτό επιτυγχάνεται με την προσομοίωση πραγματικού κώδικα επεξεργαστή σε επίπεδο κύκλων CPU (κώδικα μηχανής) έναντι της προσομοίωσης μοντέλων δικτύου που χρησιμοποιείται από παρόμοια λογισμικά (TOSSIM [27], ATEMU [28]). Η υλοποίηση του AVRORA έχει γίνει σε γλώσσα Java, πράγμα που ενισχύει τη φορητότητα (λειτουργικού συστήματος και γλώσσας υλοποίησης) και την ευελιξία σε σχέση με τα προαναφερόμενα λογισμικά που είναι γραμμένα σε κώδικα C.

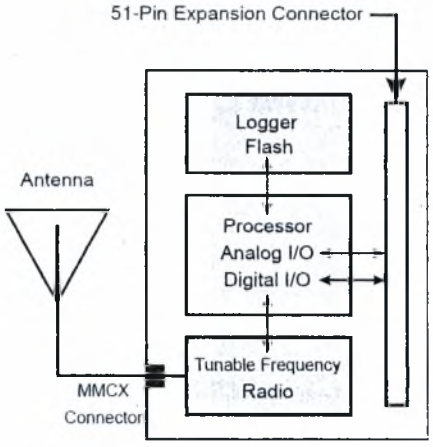
6.2 Πλατφορμα Crossbow MICA2

Η προσομοίωση έχει γίνει ειδικά για συσκευές hardware, πλατφόρμας mica2/mica2dot [29] (Σχήμα 6.2) της εταιρείας Crossbow. Πρόκειται για τη δημοφιλέστερη πλατφόρμα τρίτης γενιάς ασύρματων δικτύων αισθητήρων. Τα βασικότερα χαρακτηριστικά της είναι:

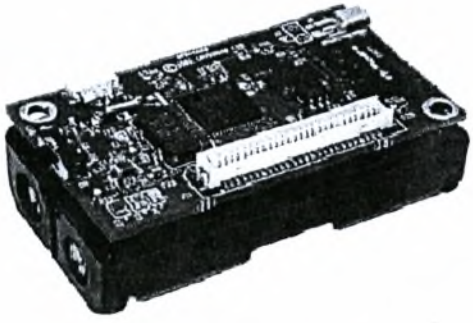
- 868/916, 433 ή 310 MHz Multi-Channel Radio Transceiver
- 38.4 kbps data rate radio
- Atmel ATmega128L μικροεπεξεργαστής
- 128 Kb program flash memory
- 512 Kb serial flash για τις μετρήσεις
- 18 γρ. συνολικό βάρος

- 2 AA μπαταρίες

Επιπρόσθετα, το MICA2 έχει ενσωματωμένο ένα 51-pin connector ο οποίος επιτρέπει τη σύνδεση με ένα πλήθος περιφερειακών. Υποστηρίζει διεπαφές Analog Input, Digital I/O, I2C, SPI και UART. Περισσότερα χαρακτηριστικά της πλατφόρμας αναγράφονται στο Παράρτημα Β.



Σχήμα 6.1 Διάγραμμα πλατφόρμας MPR400CB.



Σχήμα 6.2 Mica2 Mote.

6.3 Περιγραφη Εφαρμογης και Μεθοδολογια

6.3.1 Υπόθεση Εργασίας

Για την εκτίμηση της απόδοσης του πρωτοκόλλου, υλοποιήθηκε δίκτυο αποτελούμενο από σταθμό βάσης (BS) συνδεδεμένο με τη σειριακή θύρα (UART) ενός Η/Υ και δύο clients (Σχήμα 6.3) οι οποίοι συλλέγουν τιμές από τον προσαρμοσμένο αισθητήρα τους και αναφέρουν τα αποτελέσματα στο σταθμό βάσης.

Σχήμα 6.3 Υπόθεση Εργασίας Εφαρμογής.

Για την επικοινωνία του BS με τους αισθητήρες έχει υλοποιηθεί εφαρμογή γραμμένη σε Java, η οποία εκτελείται στον Η/Υ, επιτρέποντας την αποστολή πακέτων (packet injection) προς το δίκτυο. Τα πακέτα αυτά παίζουν το ρόλο εντολών προς τους αισθητήρες (π.χ. έναρξη/τερματισμός λειτουργίας).

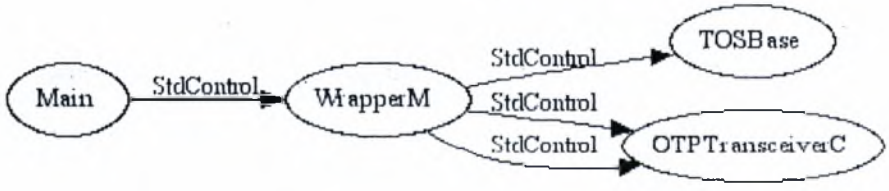
Παράλληλα, για την προβολή των δεδομένων που συλλέγονται από τον ADC στην οθόνη, γίνεται χρήση μιας δεύτερης Java εφαρμογής (Oscilloscope). Η εφαρμογή αυτή εκτελείται στον Η/Υ και ακούει για εισερχόμενα πακέτα, από τη σειριακή θύρα, προβάλλοντας τις τιμές σε γράφημα συναρτήσει του χρόνου.

6.3.2 Περιγραφή των Components

Παρακάτω, γίνεται μια σύντομη αναφορά στα κυριότερα στοιχεία του κώδικα προσομοίωσης και περιγράφονται τα components που χρησιμοποιήθηκαν σε αυτά, αλλά και ο τρόπος με τον οποίο έγινε η σύνδεσή τους για να παραχθεί η τελική εφαρμογή.

6.3.2.1 Wrapper

Το configuration αρχείο *Wrapper*, συνδέει τις δύο εφαρμογές *TOSBase* και *OTPTransceiver*. Χρησιμοποιείται μόνο για τις ανάγκες της προσομοίωσης σε περιβάλλον TOSSIM και Avrora, ώστε να παραχθεί ένα αυτόνομο εκτελέσιμο main.exe (για τον προγραμματισμό των αισθητήρων δεν είναι απαραίτητο). Ειδικότερα, στο αρχείο αυτό γίνεται η σύνδεση (wiring) του `Main.StdControl` της εφαρμογής με το implementation file (module) *WrapperM*. Όπως φαίνεται και στο Σχήμα 6.4, το *WrapperM* με τη σειρά του συνδέει τα `StdControl` του σταθμού βάσης *TOSBase* και του αρχείου κώδικα των clients *OTPTranceiverC*, τα οποία περιγράφονται στις επόμενες παραγράφους.

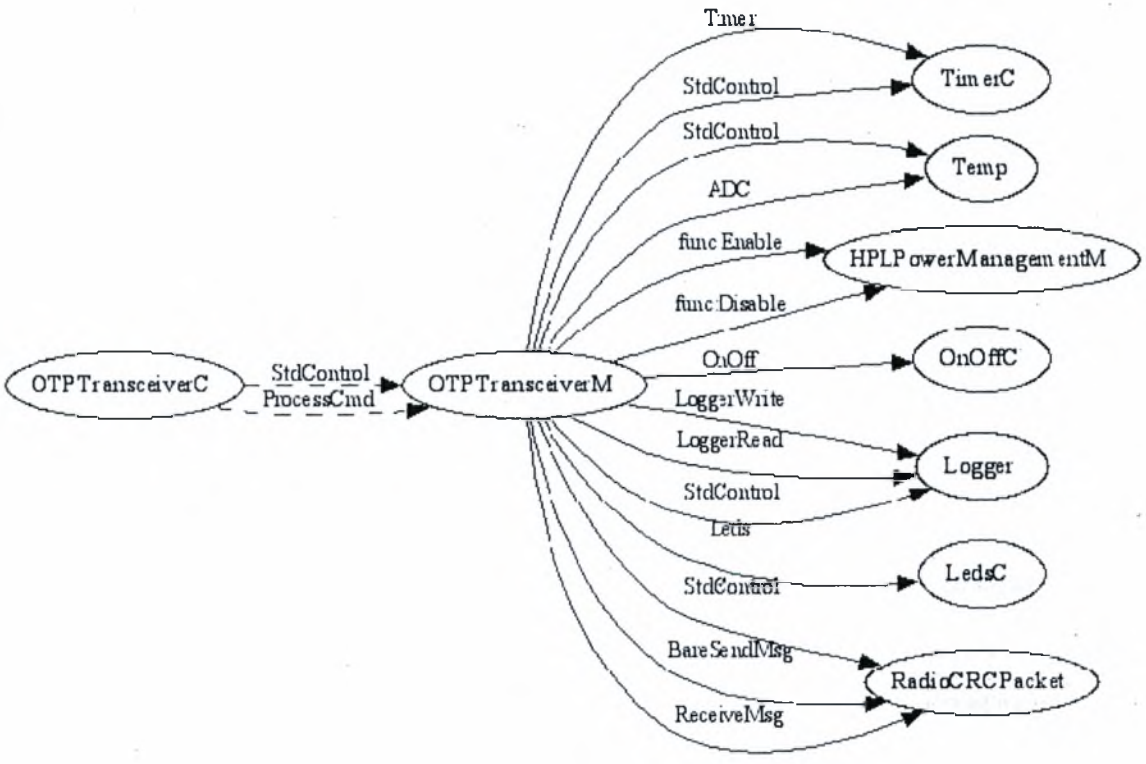


Σχήμα 6.4 Διάγραμμα ροής για τη σύνδεση των εφαρμογών TOSBase και OTPTransceiverC.

6.3.2.2 OTPTransceiverC

Το configuration αρχείο *OTPTransceiverC* περιέχει τον κώδικα που συνδέει τα components του client. Στο Σχήμα 6.5 που ακολουθεί, παρατηρούμε ότι στο module *OTPTransceiverM*, εκτός από το interface *StdControl* που απαιτείται για αρχικοποίηση, έναρξη και τερματισμό της εφαρμογής, υλοποιείται και το interface *ProcessCmd* το οποίο εκετελεί τις εντολές που λαμβάνει ο client από τον BS. Τα κυριότερα components που χρησιμοποιήθηκαν είναι τα εξής:

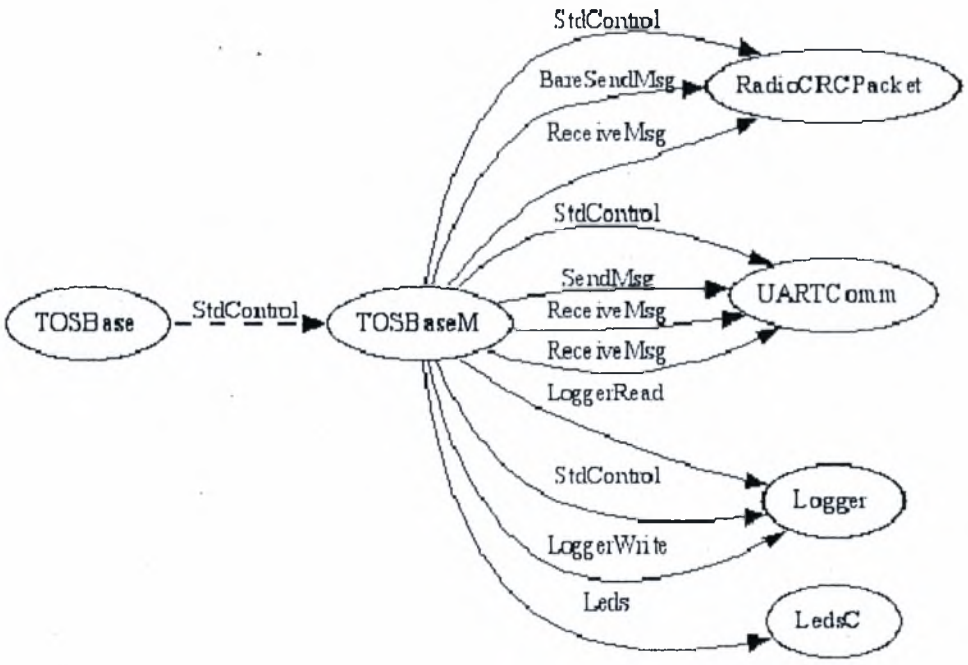
- *TimerC*. Ο χρονιστής του προγράμματος, βάσει του οποίου λαμβάνονται τιμές από τον ADC κάθε x milliseconds.
- *Temp*. Παράδειγμα αισθητήρα, ο οποίος λαμβάνει τιμές για διάφορα επίπεδα θερμοκρασίας.
- *HPLPowerManagementM*. Module που διαχειρίζεται την ενεργειακή κατάσταση στην οποία βρίσκεται ο client. Όταν π.χ. παραμείνει για αρκετή ώρα σε κατάσταση idle μεταβαίνει σε κατάσταση sleep.
- *OnOffC*. Άμεσα συσχετιζόμενο με το HPLPowerManagementM. Υλοποιεί τις εντολές on-off, θέτοντας το RF σε κατάσταση sleep/awake.
- *Logger*. Η EEPROM μνήμη των αισθητήρων
- *LedsC*. Ρυθμίζει τη συμπεριφορά των τριών ενσωματωμένων Leds (κόκκινο, πράσινο, κίτρινο).
- *RadioCRCPacket*. Υλοποιεί το module CC1000 radio με το οποίο είναι εφοδιασμένοι οι αισθητήρες Mica2. Λειτουργεί στη συσχνότητα 916MHz και περιλαμβάνει τις ρουτίνες αποστολής και λήψης με παράλληλο έλεγχο ακεραιότητας των πακέτων (με CRC).



Σχήμα 6.5 Διάγραμμα ροής για τον client OTPTransceiver.

6.3.2.3 TOSBase

Τέλος, το αρχείο *TOSBase*, υλοποιεί τη σύνδεση των components που χρησιμοποιεί το module *TOSBaseM* για τον κώδικα του σταθμού βάσης (Σχήμα 6.6). Εκτός από τα components *RadioCRCPacket, Logger* και *LedsC*, που είδαμε παραπάνω γίνεται χρήση και του component *UARTComm*, το οποίο είναι υπεύθυνο για την επικοινωνία του σταθμού βάσης με τον SerialForwarder του TinyOS μέσω των εντολών `SendMsg`, `ReceiveMsg`.



Σχήμα 6.6 Διάγραμμα ροής για τον σταθμό βάσης TOSBase.

6.3.3 Εκτέλεση Προγράμματος

Για την εκτέλεση του προγράμματος απαιτείται να υπάρχει εγκατεστημένο το πρόγραμμα TinyOS-1.0.x σε περιβάλλον Linux ή Cygwin. Για τη συγκεκριμένη προσομοίωση προτιμήθηκε το Cygwin καθώς υπάρχει καλύτερη υποστήριξη από την κοινότητα του TinyOS (tinyos-help-bounces@Millennium.Berkeley.EDU).

Αφού τοποθετηθεί ο φάκελος του προγράμματος στον αντίστοιχο φάκελο εφαρμογών /tinyos-1.x/apps/OneTimePad, ενεργοποιούμε τον debugger του TinyOS με την εντολή:

```
$ export DBG=usr1,boot,logger
```

και στη συνέχεια, δημιουργούμε το εκτελέσιμο του προγράμματος με τις παρακάτω εντολές:

```
$ make pc (για προσομοίωση στο περιβάλλον TOSSIM)
```

```
$ ./build/pc/main.exe -b=0 3
```

Η προαιρετική παράμετρος *b* καθορίζει το χρόνο εκκίνησης (boot time) των motes. Με τιμή 0 επιλέγουμε ταυτόχρονη εκκίνηση όλων.

Όταν το πρόγραμμα εκκινείται, κάθε κόμβος καταγράφει την pad στη EEPROM μνήμη του. Στη συνέχεια, ο σταθμός βάσης στέλνει μήνυμα αρχικοποίησης στο οποίο οι κόμβοι απαντούν με τον τρόπο που είδαμε στο Κεφάλαιο 5. Για το σκοπό αυτό, σε ένα δεύτερο παράθυρο κονσόλας, τρέχουμε τον SerialForwarder του TinyOS με την εντολή:

```
$ java net.tinyos.sf.SerialForwarder -comm tossim-serial@localhost
```

και στη συνέχεια, σε τρίτο παράθυρο, αφού μεταβούμε εντός του φακέλου της εφαρμογής OneTimePad δίνουμε την εντολή:

```
$ java Inject probe
```

Αφού ολοκληρωθεί η αρχικοποίηση και τοποθετηθούν οι αυθεντικοποιημένοι κόμβοι στο ευρετήριο, κάθε κόμβος αναμένει εντολή από το σταθμό βάσης. Η εντολή αυτή δίνεται από το ίδιο παράθυρο με *start [npackets nsamples interval_ms].* Για παράδειγμα,

```
$ java Inject start 10 10 200
```

Με την εντολή start, αποστέλεται πακέτο μέσω σειριακής προς τον σταθμό βάσης, ο οποίος με τη σειρά του, το προωθεί στους κόμβους του δικτύου με broadcasting. Η πρώτη παράμετρος αφορά το πλήθος των πακέτων, η δεύτερη το πλήθος δειγμάτων του ADC που επιθυμούμε να περιέχει κάθε πακέτο και η τελευταία το διάστημα (σε ms) που μεσολαβεί μεταξύ των δειγμάτων που συλλέγει ο αισθητήρας.

Άλλες εντολές που έχουν υλοποιηθεί είναι η *stop* και *on/off*.

```
$ java Inject (stop || on || off)
```

6.3.4 Στιγμιότυπο Εκτέλεσης

Πριν δούμε αναλυτικά ένα στιγμιότυπο εκτέλεσης της προσομοίωσης, κρίνεται αναγκαία η περιγραφή της δομής του πακέτου που χρησιμοποιεί το TinyOS για την επικοινωνία μεταξύ των κόμβων. Στον Πίνακας 6.1 φαίνεται η μορφή του πακέτου και στον Πίνακας 6.2 δίνεται μια μικρή

περιγραφή των πεδίων του. Στο πεδίο data αποθηκεύονται οι μεταβλητές *source_id* για το αναγνωριστικό του κόμβου, η θέση του κλειδιού στην Pad (*key_pos*) και τα κρυπτογραφημένα δεδομένα (*payload*) που συλλέγονται από τον αισθητήρα (βλέπε §B.3.2.4).

Πίνακας 6.1 Το struct TOS_Msg του TinyOS.

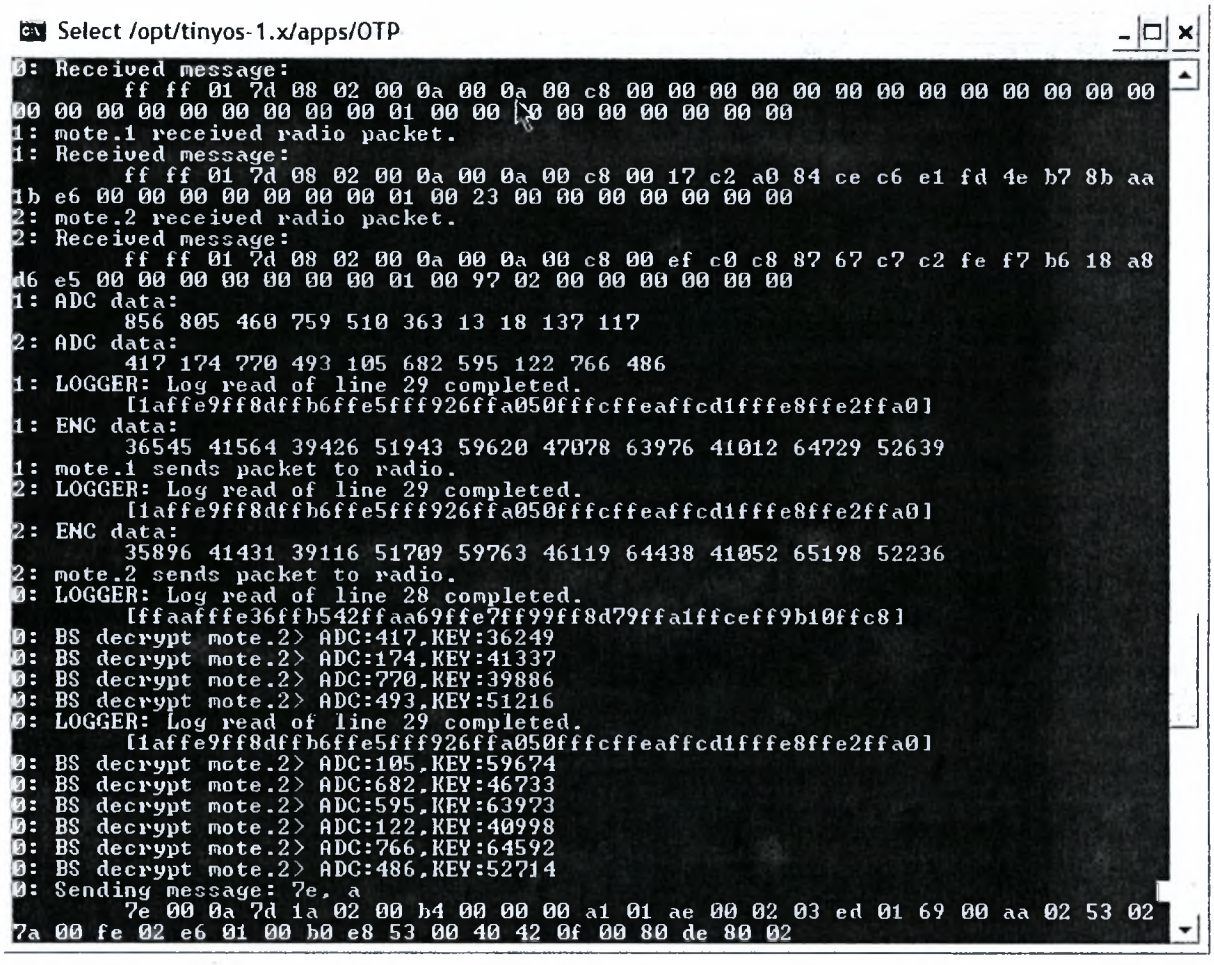
```
typedef struct TOS_Msg
{
  /* The following fields are transmitted/received on the radio. */
  uint16_t addr;
  uint8_t type;
  uint8_t group;
  uint8_t length;
  int8_t data[TOSH_DATA_LENGTH];
  uint16_t crc;

  /* The following fields are not actually transmitted or received
   * on the radio! They are used for internal accounting only.
   * The reason they are in this structure is that the AM interface
   * requires them to be part of the TOS_Msg that is passed to
   * send/receive operations.
   */
  uint16_t strength;
  uint8_t ack;
  uint16_t time;
  uint8_t sendSecurityMode;
  uint8_t receiveSecurityMode;
} TOS_Msg;
```

Πίνακας 6.2 Περιγραφή Πεδιών Πακέτου TinyOS.

Address		Message Type	Group ID	Data Length	Data			CRC	
0	1	2	3	4	5	...	n-2	n-1	n

Byte #	Πεδίο	Περιγραφή
0-1	Message Address	☐ Broadcast Address (0xFFFF) ☐ UART Address (0x007e) ☐ Node Address
2	Message Type	Active Message (AM) - Μοναδικό ID του τύπου του μηνύματος. Τυπικά, κάθε εφαρμογή έχει το δικό της AM type.
3	Group ID	Μοναδικό ID της ομάδας στην οποία ανοίκει κάθε κόμβος του δικτύου. Η προεπιλεγμένη τιμή είναι 125 (0x7d). Μόνο κόμβοι με ίδιο group ID μιλούν μεταξύ τους.
4	Data Length	Το μήκος (l) σε bytes του data payload. Δεν περιέχει το CRC ή frame synch bytes. Προεπιλεγμένο μέγεθος 29 bytes.
5 ...n-2	Payload data	Το πραγματικό περιεχόμενο του μηνύματος μήκους l
n-1, n	CRC	(2) bytes για τη διασφάλιση της ακεραιότητας του μηνύματος.

Αφού δώσουμε την εντολή start στο δίκτυο, οι κόμβοι αρχίζουν να συλλέγουν τιμές τις οποίες αναφέρουν στο σταθμό βάσης. Η παραπάνω διαδικασία, φαίνεται στο παράθυρο εκτέλεσης της εφαρμογής με τη βοήθεια των μηνυμάτων του DBG.



Σχήμα 6.7 Στιγμιότυπο εκτέλεσης του προγράμματος σε περιβάλλον TOSSIM.

Στο Σχήμα 6.7, φαίνονται με λεπτομέρεια, τα μηνύματα που μεταδίδονται από το σταθμό βάσης προς τους κόμβους. Αφού ο *BS* (με id 0) λάβει την εντολή εκκίνησης από την UART, προωθεί το πακέτο στους κόμβους 1 και 2 (στη broadcast διεύθυνση 0xffff). Το παρακάτω μήνυμα, για παράδειγμα, προέρχεται από την UART και περιέχει δεδομένα 8 bytes με data payload `02 00` (για τον τύπο της εντολής start) και `0a 00, 0a 00, c8 00` (για τις τιμές 10, 10, 200). Σημειώνεται ότι τα δεδομένα του πακέτου είναι γραμμένα σε little-endian format και διαβάζονται από δεξιά προς αριστερά. Για παράδειγμα, η διεύθυνση της UART 0x007e εμφανίζεται με το λιγότερο σημαντικό byte πρώτο, ως 0x7e00.

```
2: mote.2 received radio packet.
2: Received message:
        ff ff 01 7d 08 02 00 0a 00 0a 00 c8 00 ef c0 c8 87 67 c7 c2 fe f7
b6 18 a8 d6 e5 00 00 00 00 00 00 00 01 00 97 02 00 00 00 00 00 00
```

Αφού λάβουν το μήνυμα και οι δύο κόμβοι, παίρνουν τις 10 τιμές (ADC data) που τους ζητήθηκε και διαβάζουν από την EEPROM (LOGGER) όσα κλειδιά χρειάζονται για την κρυπτογράφηση του μηνύματος. Όπως φαίνεται και στο Σχήμα 6.7, η EEPROM είναι δομημένη σε γραμμές των 16 bytes, και κάθε κλήση ανάγνωσης διαβάζει μια ολόκληρη γραμμή. Στη συνέχεια, το μήνυμα κρυπτογραφείται και αποστέλεται στο σταθμό βάσης.

```
2: ADC data:
        417 174 770 493 105 682 595 122 766 486
1: LOGGER: Log read of line 29 completed.
        [1affe9ff8dffb6ffe5fff926ffa050fffcffeaffcd1fffe8ffe2ffa0]
2: ENC data:
        35896 41431 39116 51709 59763 46119 64438 41052 65198 52236
2: mote.2 sends packet to radio.
```

Όταν ο *BS* λάβει το μήνυμα, εκτελεί την αντίστροφη διαδικασία κατά την οποία αποκρυπτογραφείται το μήνυμα και προωθείται προς την UART (διεύθυνση 7e 00) για την προβολή των τιμών στην εφαρμογή Oscilloscope (AM type=0a) την οποία θα δούμε παρακάτω.

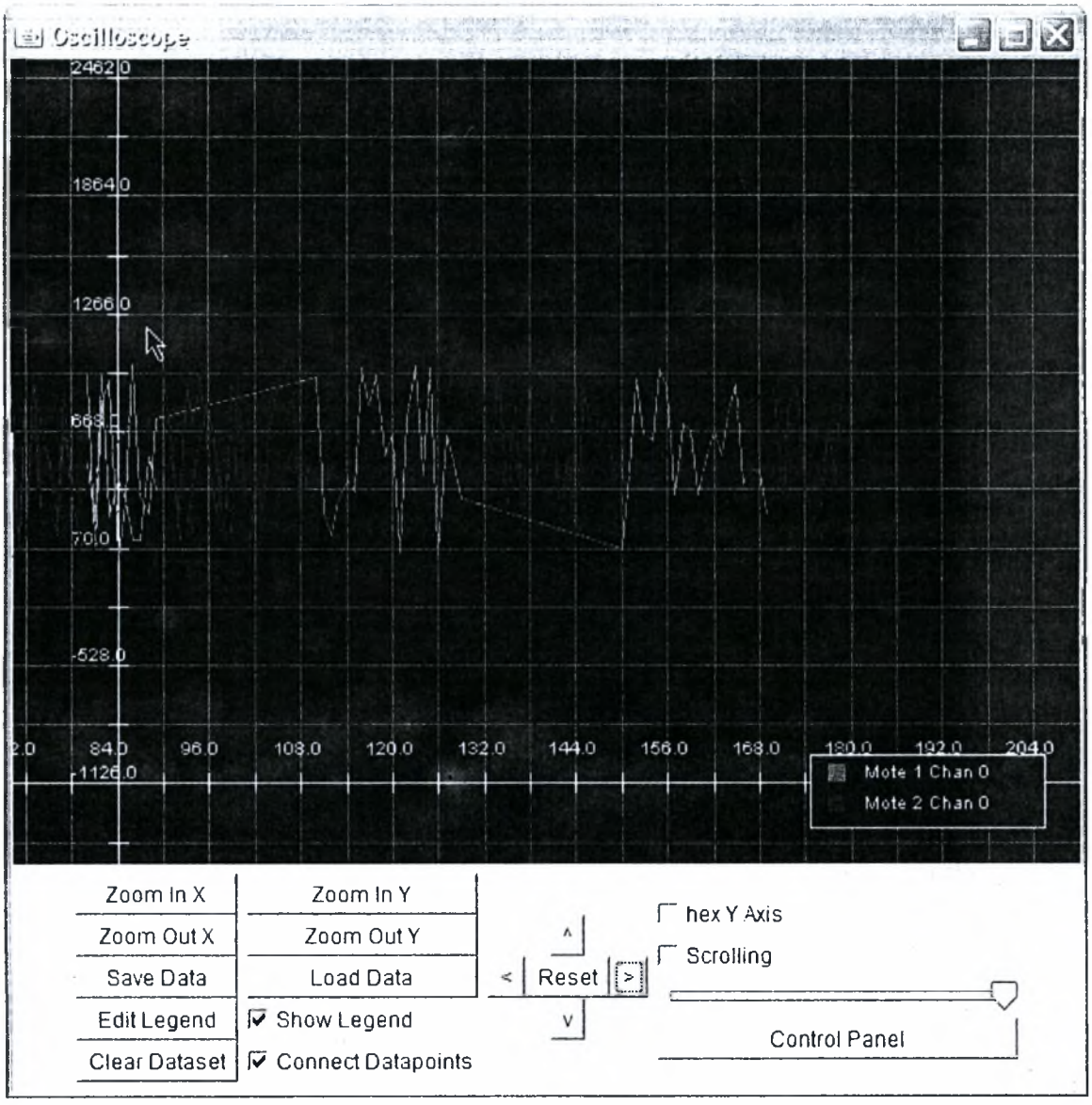
```
0: LOGGER: Log read of line 28 completed.
        [ffaafffe36ffb542ffaa69ffe7ff99ff8d79ffa1ffceff9b10ffc8]
0: BS decrypt mote.2> ADC:417,KEY:36249
0: BS decrypt mote.2> ADC:174,KEY:41337
0: BS decrypt mote.2> ADC:770,KEY:39886
0: BS decrypt mote.2> ADC:493,KEY:51216
0: LOGGER: Log read of line 29 completed.
        [1affe9ff8dffb6ffe5fff926ffa050fffcffeaffcd1fffe8ffe2ffa0]
0: BS decrypt mote.2> ADC:105,KEY:59674
0: BS decrypt mote.2> ADC:682,KEY:46733
0: BS decrypt mote.2> ADC:595,KEY:63973
0: BS decrypt mote.2> ADC:122,KEY:40998
0: BS decrypt mote.2> ADC:766,KEY:64592
0: BS decrypt mote.2> ADC:486,KEY:52714
0: Sending message: 7e, a
        7e 00 0a 7d 1a 02 00 b4 00 00 00 a1 01 ae 00 02 03 ed 01 69 00 aa
02 53 02 7a 00 fe 02 e6 01 00 b0 e8 53 00 40 42 0f 00 80 de 80 02
```

6.3.5 Oscilloscope

Παράλληλα με το debugging (μόνο για προσομοίωση σε TOSSIM), υπάρχει η δυνατότητα απεικόνισης των δειγμάτων που συλλέγονται από τον ADC σε γράφημα, συναρτήσει του χρόνου (Σχήμα 6.8). Αυτό επιτυγχάνεται με κατάλληλο wiring του σταθμού βάσης με την εφαρμογή

Oscilloscope ή οποία ακούει για πακέτα σε συγκεκριμένο AM type (10) της UART. Η εντολή για την έναρξη του Oscilloscope είναι:

```
$ java net.tinyos.oscope.oscilloscope
```



Σχήμα 6.8 Απεικόνιση των Τιμών του ADC με την Εφαρμογή Oscilloscope.

6.3.6 Λήψη Μετρήσεων με το Εργαλείο Avrora

Όπως αναφέρθηκε παραπάνω, το πρόγραμμα που χρησιμοποιήθηκε για τη διεξαγωγή των μετρήσεων είναι το Avrora. Η τρέχουσα έκδοσή του είναι η beta-1.6.0 και παρέχεται σε αυτόνομο

αρχείο jar, το οποίο τοποθετούμε στη θέση tinyos-1.x/tools/java/jars/. Παρακάτω, περιγράφεται συνοπτικά η διαδικασία που ακολουθείται για τη λήψη των μετρήσεων.

Αρχικά, στο φάκελο της εφαρμογής, εκτελούμε το Makefile, αυτή τη φορά για πλατφόρμα mica2, για να παραχθεί το εκτλέσιμο *main.exe*. Στη συνέχεια, με την εφαρμογή avr-objdump παράγουμε τον κώδικα μηχανής που αντιστοιχεί στο εκτελέσιμο που δημιουργήσαμε προηγουμένως.

```
$ make mica2
$ avr-objdump -zhD ./build/mica2/main.exe > main.od
```

Για την εκτέλεση του παραγόμενου αρχείου main.od από το Avrora, εκτελείται η ακόλουθη εντολή από το root της εφαρμογής κάνοντας προηγουμένως alias το `avrora-beta-1.6.0.jar`:

```
$ alias avrora='java -jar
c:/cygwin/opt/tinyos-1.x/tools/java/jars/avrora-beta-1.6.0.jar'
$ avrora -config-file=config.txt -seconds=20 main.od > main.txt
```

Το αρχείο config.txt περιέχει μια λίστα από παραμέτρους για τη ρύθμιση της προσομοίωσης του Avrora. Επίσης, η διάρκεια της προσομοίωσης που καθορίζεται από την παράμετρο *-seconds* πρέπει να ελεγχθεί ώστε να μην υπερβαίνει κατά πολύ την αποστολή του τελευταίου πακέτου.

```
colors=false
banner=false
license=false
status=false
input=objdump
action=simulate
simulation=sensor-network
platform=mica2
nodecount=3
report-seconds
seconds-precision=2
monitors=energy,sleep,packet,serial,energy-profile
```

Ακολούθως, εκκινούμε το SerialForwarder του TinyOS με την εντολή:

```
$ java net.tinyos.sf.SerialForwarder -comm network@localhost:2390
```

και δίνουμε την εντολή start στο δίκτυο. Σημειώνεται ότι για τις μετρήσεις, το μήνυμα αυθεντικοποίησης αποστέλεται αυτόματα από τον σταθμό βάσης και δεν χρειάζεται η εντολή probe.

```
$ java Inject start 10 10 200
```

Οι μετρήσεις για την κατανάλωση ενέργειας και για πλήθος άλλων χαρακτηριστικών της προσομοίωσης περιέχονται στο αρχείο εξόδου του Avrora main.txt.

6.4 Εκτιμηση Αποδοσης Πρωτοκολλου

Ένα πρωτόκολλο ασφάλειας για να είναι αποδοτικό πρέπει να παρέχει τις παρακάτω υπηρεσίες:

- Ασφαλή αρχικοποίηση και ασφαλή ανταλλαγή κλειδιών σε όλη τη διάρκεια της επιχειρησιακής λειτουργίας του.
- Επεκτασιμότητα - Προσαρμοστικότητα: Πρέπει να παρέχει εύκολη προσθήκη και διαγραφή κόμβων.
- Την υποστήριξη της ασφαλούς λειτουργίας του δικτύου σε μη προκαθορισμένα περιβάλλοντα, και τέλος
- Την απαγόρευση χρήσης της υποδομής του σε μη εξουσιοδοτημένους κόμβους.

Το πρωτόκολλο SecureMem/OTP παρέχει τις παραπάνω υπηρεσίες και επιπλέον εμφανίζει πολύ καλά χαρακτηριστικά ως προς την αντοχή του στην κρυπτανάλυση και στην κατανάλωση ενέργειας. Στις επόμενες παραγράφους θα εξετάσουμε αυτά τα χαρακτηριστικά.

6.4.1 Αντοχή στην Κρυπτανάλυση

Η αντοχή του πρωτοκόλλου στην κρυπτανάλυση στηρίζεται στην ισχύ του αλγόριθμου κρυπτογράφησης OTP, ο οποίος είναι ο μόνος αδιάσπαστος αλγόριθμος με μαθηματική απόδειξη (βλέπε Κεφάλαιο ΜΕΡΟΣ Ι -3). Επιπλέον, η ισχυρή προστασία του κόμβου ενάντια σε προσπάθειες παραβίασής του, εξασφαλίζει τη φυσική του ασφάλεια, και ιδιαίτερα την ασφάλεια του κρυπτογραφικού του υλικού.

Η χρήση ευρετηρίου κόμβων σε επίπεδο δικτύου και ομάδας, εξασφαλίζει την αυθεντικοποίηση των πακέτων μιας και κανένα μήνυμα δεν τυγχάνει επεξεργασίας από τον *BS* ή τον *Ch* εάν ο αποστολέας του δεν είναι καταχωρημένος σε ένα από αυτά τα ευρετήρια.

Όπως έχει ήδη αναφερθεί, τα κλειδιά που χρησιμοποιούνται από τον αλγόριθμο OTP πρέπει να είναι τουλάχιστον ίσα με το μέγεθος του μηνύματος που κρυπτογραφείται. Για το λόγο αυτό έχουν χρησιμοποιηθεί κλειδιά μεγέθους 2 bytes.

6.4.2 Κατανάλωση Ενέργειας

Για την εκτίμηση της απόδοσης του SecureMem/OTP έγινε σύγκριση με το πρωτόκολλο TinySec και OTP/TinySec· ένα συνδυασμό των πρωτοκόλλων SecureMem/OTP και TinySec. Στην πρώτη περίπτωση, εφαρμόζεται κρυπτογράφηση με τον αλγόριθμο SkipJack, ο οποίος σύμφωνα με το [14] είναι ο καταλληλότερος αλγόριθμος (μαζί με τον RC5) για software υλοποίηση σε embedded μικροεπεξεργαστές. Ο SkipJack λειτουργεί σε CBC-Mode με ένα μοναδικό κλειδί για όλο το δίκτυο, το οποίο είναι τοποθετημένο από πριν σε κάθε κόμβο, και αυθεντικοποίηση με χρήση Message Authentication Code (MAC). Στην περίπτωση OTP/TinySec, η κρυπτογράφηση γίνεται με τον αλγόριθμο OTP, ενώ για την αυθεντικοποίηση χρησιμοποιείται MAC (σε TINYSEC_AUTH_ONLY mode [30]).

Πίνακας 6.3 Χαρακτηριστικά Πρωτοκόλλων Προσομοίωσης.

	Κρυπτογράφηση	**Αυθεντικοποίηση**
SecureMem/OTP	One Time Pad	Ευρετήριο Κόμβων
TinySec	SkipJack	MAC
OTP/TinySec	One Time Pad	MAC

Τα χαρακτηριστικά που επηρεάζουν την ενεργειακή απόδοση ενός αλγορίθμου κρυπτογράφησης είναι η ενέργεια που καταναλώνει το υποσύστημα επεξεργασίας στην ενεργό περιοχή λειτουργίας του (active state). Για το λόγο αυτό, οι μετρήσεις έχουν ληφθεί με γνώμονα τη μεγιστοποίηση της συχνότητας λειτουργίας των κόμβων, η οποία καθορίζεται από την περίοδο του ρολογιού κατά την οποία λαμβάνονται οι τιμές από τον ADC.

Για τον έλεγχο απόδοσης του αλγορίθμου, λαμβάνονται μετρήσεις κατανάλωσης ενέργειας με τη βοήθεια του εργαλείου Avrora, για μεταβλητό πλήθος (10, 20, 50, 100) και μεταβλητά μεγέθη (2, 4, 10, 20 bytes) πακέτων. Για το σκοπό αυτό, στέλνονται στο δίκτυο πακέτα συνολικού οφέλιμου φορτίου (data payload) 200 bytes. Οι μετρήσεις επαναλαμβάνονται για διαφορετικούς ρυθμούς ρολογιού (clock rate 80,100), σε κάθε χτύπο του οποίου λαμβάνεται μέτρηση από τον ADC.

Οι παρακάτω πίνακες συγκεντρώνουν τα αποτελέσματα ως προς την κατανάλωση ενέργειας εκάστου πρωτοκόλλου. Το συνολικό οφέλιμο φορτίο προκύπτει από το γινόμενο των πεδίων ADC Data και Total Packets, το οποίο είναι πάντοτε 200 bytes. Δεδομένου ότι κάθε τιμή του ADC είναι 2 bytes, το πεδίο *ADC Data* εκφράζει έμμεσα το πλήθος τιμών του ADC, π.χ. θερμοκρασίας, που

περιέχει κάθε πακέτο που αποστέλλεται στο δίκτυο. Κάθε τιμή συλλέγεται από τον ADC με ρυθμό ρολογιού *Clock Rate*. Η προσομοίωση ολοκληρώνεται και λαμβάνονται τα συγκεντρωτικά αποτελέσματα όταν αποσταλούν όλα τα πακέτα που αναγράφονται στο πεδίο *Total Packets*.

Τέλος, στο πεδίο *Active CPU Cycles* καταγράφεται και το ποσοστό των κύκλων που η CPU παραμένει σε ενεργή κατάσταση (active state) σε σχέση με το συνολικό αριθμό κύκλων που απαιτούνται για την προσομοίωση.

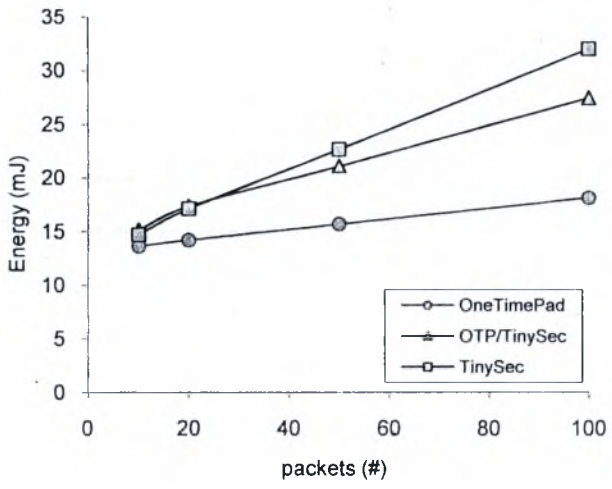
6.4.3 Clock Rate=80 ms, Total Payload=200 bytes

6.4.3.1 Μετρήσεις Απόδοσης Κόμβων

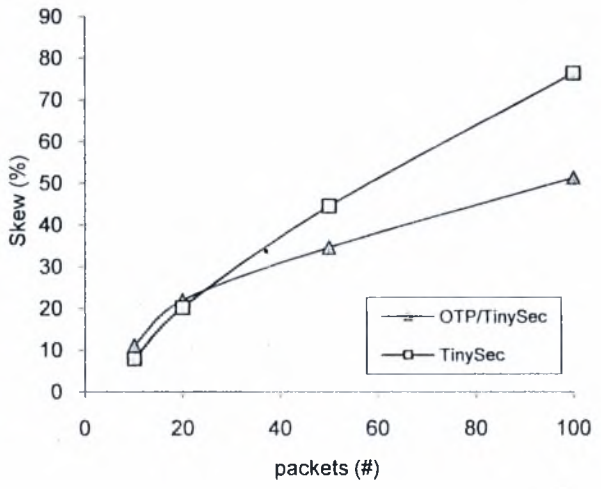
Πίνακας 6.4 Client, Clock Rate=80ms, Total Payload=200 bytes.

Clock Rate	Total Packets	ADC Data	Active CPU Cycles (%)			Energy (mJ)		
(ms)	(#)	(bytes)	OTP	TinySec	OTP/TinySec	OTP	TinySec	OTP/TinySec
80	100	2	9,85	16,97	13,42	18,12	31,97	27,42
80	50	4	8,52	12,02	11,27	15,66	22,62	21,08
80	20	10	7,74	9,43	9,10	14,24	17,12	17,36
80	10	20	7,42	8,76	8,25	13,64	14,72	15,16

Η ενεργειακή απόδοση των τριών πρωτοκόλλων φαίνεται στα γραφήματα που ακολουθούν. Στον οριζόντιο άξονα και των δύο σχημάτων παριστάνεται το πλήθος των πακέτων που αποστέλλονται από τους κόμβους πελάτες προς τον σταθμό βάσης.



Σχήμα 6.9 Client, CPU Energy Consumption vs Packets (80ms).



Σχήμα 6.10 Client, CPU Energy Skew vs Packets (80ms).

Το Σχήμα 6.9 δείχνει τη συνολική ενέργεια σε mJ που απαιτείται από κάθε πρωτόκολλο για τη μετάδοση συγκεκριμένου πλήθους πακέτων. Παρατηρούμε ότι η μικρότερη κλίση εμφανίζεται στο πρωτόκολλο OTP, πράγμα που αποδεικνύει την βέλτιστη συμπεριφορά του, σε σχέση με τα υπόλοιπα πρωτόκολλα. Το γεγονός αυτό οφείλεται στο ότι ο αλγόριθμος One Time Pad εμφανίζει πολυπλοκότητα γραμμική στο πλήθος των στοιχείων που πρόκειται να κρυπτογραφηθούν. Η μικρή επιβάρυνση του ενεργειακού κόστους καθώς τα πακέτα αυξάνονται, οφείλεται αποκλειστικά στη διαδικασία δημιουργίας τους.

Αντίθετα, η απόδοση των πρωτοκόλλων OTP/TinySec και TinySec είναι αισθητά επιβαρυμένη από τις διαδικασίες υπολογισμού των MAC και της κρυπτογράφησης SkipJack για το πρωτόκολλο TinySec. Οι διαδικασίες αυτές είναι σχετικά πολύπλοκες και όπως προκύπτει από το γράφημα είναι προτιμότερο να εφαρμόζονται σε μεγαλύτερου μήκους παρά σε μεγαλύτερου πλήθους πακέτα.

Στο Σχήμα 6.10 φαίνεται η ενεργειακή απόκλιση των πρωτοκόλλων OTP/TinySec και TinySec σε σχέση με το πρωτόκολλο OTP. Παρατηρείται ότι το TinySec εμφανίζει τον μεγαλύτερο ρυθμό απόκλισης γεγονός που οφείλεται στην πολυπλοκότητα τόσο της αυθεντικοποίησης όσο και της κρυπτογράφησης σε σχέση με το OTP/TinySec το οποίο επιβαρύνεται μόνο από την αυθεντικοποίηση.

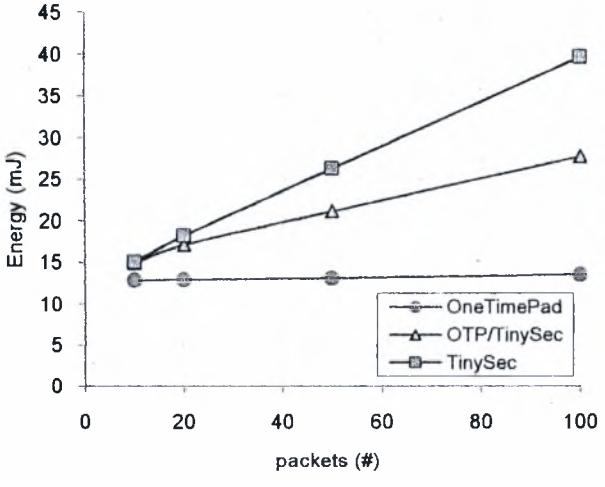
6.4.3.2 Μετρήσεις Απόδοσης Σταθμού Βάσης

Πίνακας 6.5 Base Station, Clock Rate=80ms, Total Payload=200 bytes.

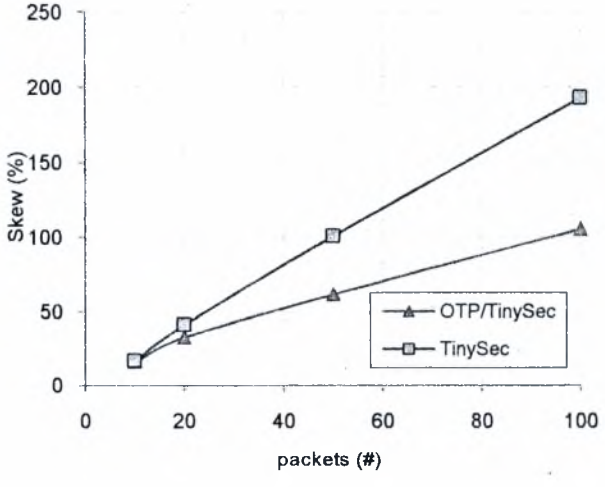
Clock Rate	Total Packets	ADC Data	Active CPU Cycles (%)			Energy (mJ)		
(ms)	(#)	(bytes)	OTP	TinySec	OTP/TinySec	OTP	TinySec	OTP/TinySec
80	100	2	7,36	21,05	13,60	13,53	39,67	27,78
80	50	4	7,13	13,97	11,31	13,11	26,32	21,16
80	20	10	7,02	10,02	8,95	12,90	18,19	17,07
80	10	20	6,99	8,93	8,10	12,85	15,00	14,89

Στα παρακάτω σχήματα, ο οριζόντιος άξονας αφορά στο πλήθος των πακέτων που λαμβάνει ο σταθμός βάσης από τους κόμβους πελάτες του δικτύου. Στο Σχήμα 6.11 αξίζει να παρατηρηθεί η σχεδόν μηδενική κλίση του πρωτοκόλλου OTP, λόγω της γραμμικής πολυπλοκότητας που αναφέρθηκε και στην περίπτωση των κόμβων πελατών. Η μικρή πρόσθετη επιβάρυνση οφείλεται αποκλειστικά στην αυθεντικοποίηση των πακέτων από τον σταθμό βάσης.

Η συμπεριφορά τέλος, των πρωτοκόλλων OTP/TinySec και TinySec, παραμένει όμοια με αυτή των κόμβων πελατών.



Σχήμα 6.11 Base Station, CPU Energy Consumption vs Packets (80ms).



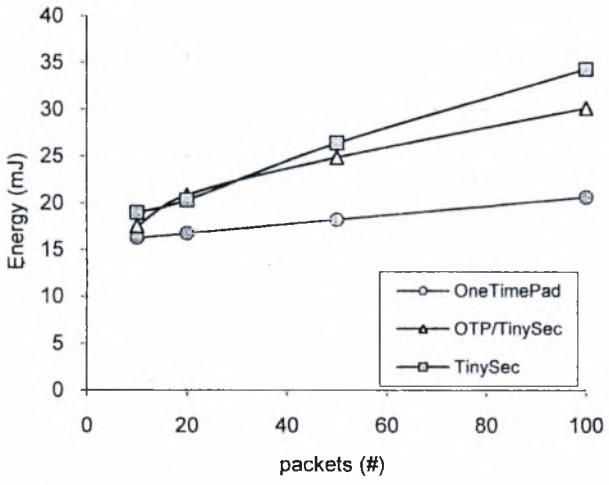
Σχήμα 6.12 Base Station, CPU Energy Skew vs Packets (80ms).

6.4.4 Clock Rate=100 ms, Total Payload=200 bytes

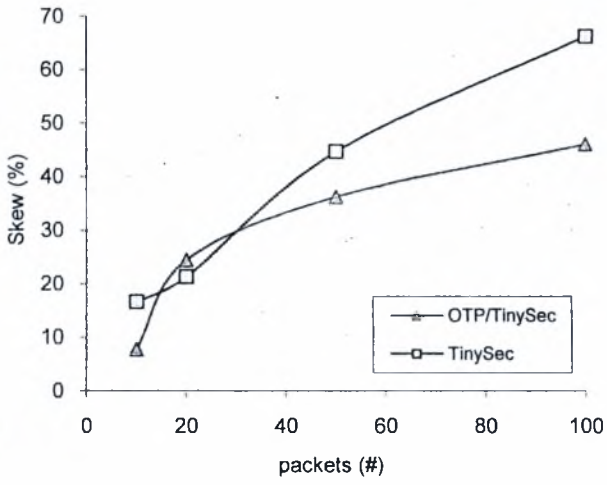
6.4.4.1 Μετρήσεις Απόδοσης Κόμβων

Πίνακας 6.6 Client, Clock Rate=100ms, Total Payload=200 bytes.

Clock Rate	Total Packets	ADC Data	Active CPU Cycles (%)			Energy (mJ)		
(ms)	(#)	(bytes)	OTP	TinySec	OTP/TinySec	OTP	TinySec	OTP/TinySec
100	100	2	9,08	15,07	12,06	20,62	34,27	30,11
100	50	4	8,03	10,56	9,,94	18,22	26,36	24,82
100	20	10	7,37	8,76	8,34	16,74	20,30	20,83
100	10	20	7,16	8,10	7,71	16,26	18,95	17,51



Σχήμα 6.13 Client, CPU Energy Consumption vs Packets (100ms).

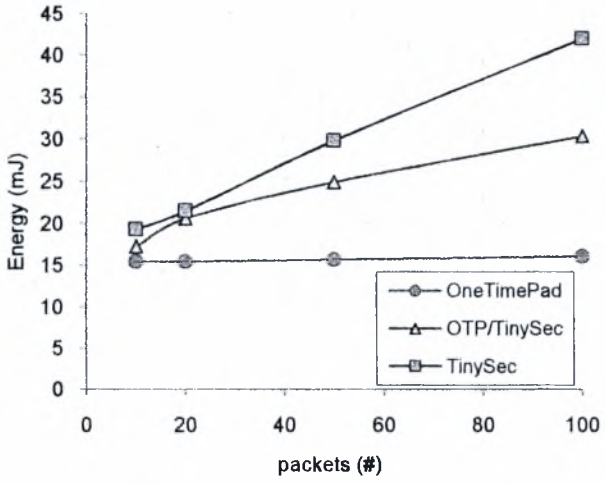


Σχήμα 6.14 Client, CPU Energy Skew vs Packets (100ms).

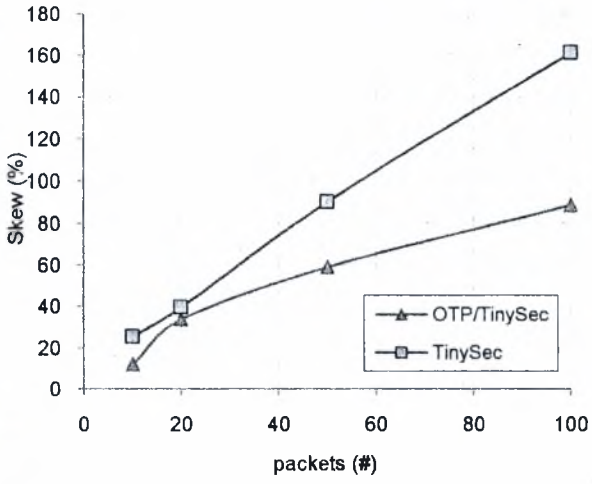
6.4.4.2 Μετρήσεις Απόδοσης Σταθμού Βάσης

Πίνακας 6.7 Base Station, Clock Rate=100ms, Total Payload=200 bytes.

Clock Rate	Total Packets	ADC Data	Active CPU Cycles (%)			Energy (mJ)		
(ms)	(#)	(bytes)	OTP	TinySec	OTP/TinySec	OTP	TinySec	OTP/TinySec
100	100	2	7,08	18,48	12,15	16,08	42,04	30,35
100	50	4	6,89	11,93	9,97	15,65	29,78	24,89
100	20	10	6,78	9,27	8,23	15,39	21,47	20,56
100	10	20	6,78	8,26	7,59	15,39	19,31	17,22



Σχήμα 6.15 Base Station, CPU Energy Consumption vs Packets (100ms).



Σχήμα 6.16 Base Station, CPU Energy Skew vs Packets (100ms).

Η σελίδα αυτή είναι σκόπιμα λευκή.

7. ΣΥΜΠΕΡΑΣΜΑΤΑ

7.1 ΠΑΡΑΤΗΡΗΣΕΙΣ - ΣΥΜΠΕΡΑΣΜΑΤΑ

Τα αποτελέσματα τα οποία προκύπτουν από την ανάλυση των παραπάνω στοιχείων είναι τα αναμενόμενα. Το πρωτόκολλο SecureMem/OTP, το οποίο κάνει χρήση του αλγόριθμου OTP, απαιτεί τη λιγότερη κατανάλωση ενέργειας, σε όλες τις περιπτώσεις. Το γεγονός αυτό εξηγείται από την απλότητα του λογικού τελεστή XOR που αυτός χρησιμοποιεί για την κρυπτογράφηση και αποκρυπτογράφηση των δεδομένων, μιας και εμφανίζει πολυπλοκότητα γραμμική στο πλήθος των στοιχείων που πρόκειται να κρυπτογραφηθούν. Τη χειρότερη συμπεριφορά εμφανίζει το πρωτόκολλο TinySec καθώς για κρυπτογράφηση χρησιμοποιεί τον αλγόριθμο SkipJack, ο οποίος είναι βελτιστοποιημένος ως προς την ταχύτητα και όχι ως προς την κατανάλωση ενέργειας. Τέλος, μεσαία συμπεριφορά παρουσιάζει το πρωτόκολλο OTP/TinySec καθώς η διαφορά με τον OTP είναι στη χρήση αυθεντικοποίησης με MAC έναντι της απλούστερης χρήσης ευρετηρίων του OTP (βλέπε Πίνακας 6.3).

Ειδικότερα, στην περίπτωση §6.4.3.1, όπου ο ρυθμός του ρολογιού είναι υψηλός, οι κόμβοι που λειτουργούν με το πρωτόκολλο SecureMem/OTP εμφανίζουν μέγιστη διαφορά ενέργειας κατά ≈76% σε σχέση με το πρωτόκολλο TinySec και κατά ≈51% σε σχέση με το πρωτόκολλο OTP/TinySec. Η διαφορά αυτή, φθίνει αντιστρόφως ανάλογα με το μέγεθος του πακέτου και το πλήθος τον μηνυμάτων. Η ίδια συμπεριφορά, όπως είναι λογικό, παρατηρείται σε όλες τις μετρήσεις διότι μειώνεται το ποσοστό των κόμβων που παραμένουν ενεργοί (σε active mode). Για τον ίδιο λόγο, τα αντίστοιχα ποσοστά με ρυθμό ρολογιού 100ms (περίπτωση §6.4.4.1), εμφανίζονται μειώμενα, σε ποσοστά ≈66% και ≈46%, για TinySec και OTP/TinySec αντίστοιχα.

Όσον αφορά τη συμπεριφορά των σταθμών βάσης (περιπτώσεις §6.4.3.2 και §6.4.4.2), η μέγιστη διαφορά στην ενέργεια που καταναλώνουν, εμφανίζεται αρκετά μεγαλύτερη κατά ≈193% και ≈105% αντίστοιχα, πράγμα που, εκτός από την αυξημένη πολυπλοκότητα, οφείλεται και στην κατανάλωση ενέργειας από τα πρωτόκολλα TinySec για την αποστολή ACK προς τους κόμβους κατά τη λήψη των μηνυμάτων.

Η σελίδα αυτή είναι σκόπιμα λευκή.

Α. ΠΑΡΑΡΤΗΜΑ - ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ MICA2/MICA2DOT

Processor/Radio Board	MPR400CB	MPR410CB	MPR420CB	Remarks
Processor Performance				
Program Flash Memory	128K bytes	128K bytes	128K bytes	
Measurement (Serial) Flash	512K bytes	512K bytes	512K bytes	> 100,000 measurements
Configuration EEPROM	4K bytes	4K bytes	4K bytes	
Serial Communications	UART	UART	UART	0-3V transmission levels
ADC	10-bit ADC	10-bit ADC	10-bit ADC	8 channel, 0-3V input
Other Interfaces	DIO, I2C, SPI	DIO, I2C, SPI	DIO, I2C, SPI	
Current Draw	8 mA	8 mA	8 mA	Active mode
	< 15 μA	< 15 μA	< 15 μA	Sleep mode
Multi-Channel Radio				
Center Frequency	868/916 MHz	433 MHz	315 MHz	ISM bands
Number of Channels	4/ 50	4	5	Programmable, country specific
Data Rate	38.4 KBaud	38.4 KBaud	38.4 KBaud	Manchester encoded
RF Power	-20 to +5 dBm	-20 to +10 dBm	-20 to +10 dBm	Programmable typical
Receive Sensitivity	-98 dBm	-101 dBm	-101 dBm	Typical, analog RSSI at AD Ch.0
Outdoor Range	500 ft	1000 ft	1000 ft	1/4 Wave dipole, line of sight
Current Draw	27 mA	25 mA	25 mA	Transmit with maximum power
	10 mA	8 mA	8 mA	Receive
	< 1 mA	< 1 mA	<1 mA	Sleep
Electromechanical				
Battery	2X AA battaries	2X AA battaries	2X AA battaries	Attached pack
External Power	2.7 - 3.3 V	2.7 - 3.3 V	2.7 - 3.3 V	Connector provided
User Interface	3 LEDs	3 LEDs	3 LEDs	User programmable
Size (mm)	58 x 32 x 7	58 x 32 x 7	58 x 32 x 7	Excluding battery pack
Weight (grams)	18	18	18	Excluding batteries
Expansion Connector	51-pin	51-pin	51-pin	All major I/O signals

Η σελίδα αυτή είναι σκόπιμα λευκή.

Β.1.1 Wrapper.nc

Το αρχείο Wrapper.nc είναι το configuration file για τη σύνδεση του interface StdControl του client OTPTransceiverC και του σταθμού βάσης TOSBase. Παρατηρούμε ότι για την εφαρμογή OTPTransceiverC δημιουργούνται δύο στιγμιότυπα (SndControl,TrdControl) του ίδιο αρχείου.

```
configuration Wrapper {}
implementation {

      components Main, WrapperM, OTPTransceiverC, TOSBase as BaseStation;

      Main.StdControl->WrapperM.StdControl;

      WrapperM.FstControl->BaseStation.StdControl;
      WrapperM.SndControl->OTPTransceiverC.StdControl;
      WrapperM.TrdControl->OTPTransceiverC.StdControl;
}
```

Β.1.2 WrapperM.nc

Πρόκειται για το module του Wrapper.nc. Εδώ γίνεται η υλοποίηση του interface StdControl των δύο εφαρμογών με εντολές init(),start() και stop(). Οι εντολές αυτές αρχικοποιούν, εκκινούν και παύουν τη λειτουργία των κόμβων. Παρατηρούμε ότι με το συγκεκριμένο wiring, ο σταθμός βάσης λαμβάνει πάντα τη διεύθυνση 0, ενώ οι κόμβοι-πελάτες τις διευθύνσεις 1 και 2.

```
module WrapperM {
        provides {
          interface StdControl;
        }
        uses {
          interface StdControl as FstControl;
          interface StdControl as SndControl;
          interface StdControl as TrdControl;
        }
      }
implementation {
        command result_t StdControl.init() {

              if (TOS_LOCAL_ADDRESS==0) {
                return (call FstControl.init());
              }
              else if ((TOS_LOCAL_ADDRESS==1)) {
                return (call SndControl.init());
              }
              else if ((TOS_LOCAL_ADDRESS==2))
                return (call TrdControl.init());
              return SUCCESS;
        }
        command result_t StdControl.start() {

               if (TOS_LOCAL_ADDRESS==0){
                 return (call FstControl.start());
               }
               else if ((TOS_LOCAL_ADDRESS==1)) {
                 return (call SndControl.start());
               }
               else if ((TOS_LOCAL_ADDRESS==2))
                 return (call TrdControl.start());

               return SUCCESS;
        }
        command result_t StdControl.stop() {

               if (TOS_LOCAL_ADDRESS==0)
                 return (call FstControl.stop());
               else if ((TOS_LOCAL_ADDRESS==1))
                 return (call SndControl.stop());
               else if ((TOS_LOCAL_ADDRESS==2))
                 return (call TrdControl.stop());

               return SUCCESS;
        }
}
```

B.2.1 OTPTransceiverC.nc

Πρόκειται για το configuration file του κώδικα λειτουργίας των κόμβων-πελατών (βλέπε §6.3.2.2 για την περιγραφή των components που χρησιμοποιούνται.

```
includes client;

configuration OTPTransceiverC {
   provides {
              interface StdControl;
              interface ProcessCmd;
   }
}
implementation {
   components
            OTPTransceiverM,
            TimerC,
            Temp as Sensor,
            RadioCRCPacket as COM,
            DisplayC,
            LedsC,
            OnOffC,
            HPLPowerManagementM as Power,
            Logger;

   StdControl = OTPTransceiverM.StdControl;
   ProcessCmd = OTPTransceiverM.ProcessCmd;

   OTPTransceiverM.Timer -> TimerC.Timer[unique("Timer")];
   OTPTransceiverM.TimerControl -> TimerC.StdControl;
   OTPTransceiverM.ADCControl -> Sensor.StdControl;
   OTPTransceiverM.ADC -> Sensor.TempADC;
   OTPTransceiverM.PowerEnable -> Power.Enable;
   OTPTransceiverM.PowerDisable -> Power.Disable;
   OTPTransceiverM.OnOff -> OnOffC;

   OTPTransceiverM.LogControl -> Logger.StdControl;
   OTPTransceiverM.LoggerWrite -> Logger.LoggerWrite;
   OTPTransceiverM.LoggerRead -> Logger.LoggerRead;

   OTPTransceiverM.Leds -> LedsC.Leds;

   OTPTransceiverM.RadioControl -> COM.Control;
   OTPTransceiverM.RadioSend -> COM.Send;
   OTPTransceiverM.RadioReceive -> COM.Receive;
}
```

Β.2.2 OTPTransceiverM.nc

Ακολουθεί η περιγραφή των κυριότερων μεθόδων του module, το οποίο υλοποιεί τη λειτουργικότητα του κόμβου-πελάτη.

Β.2.2.1 task void pad2flash()

Για τις ανάγκες τις προσομοίωσης στο περιβάλλον TOSSIM, κατά την έναρξη λειτουργίας (`StdControl.start()`), καταγράφεται το βιβλίο με τους κωδικούς στην EEPROM μνήμη κάθε κόμβου. Αυτό γίνεται με την κλήση της εντολής LoggerWrite.write(wline,ptr) μέσα από το task pad2flash(). Ο δείκτης ptr δείχνει σε πίνακα TOS_EEPROM_LINE_SIZE/2 θέσεων καθεμία μεγέθους 2 bytes.

Β.2.2.2 event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr)

Σαν αρχή της εκτέλεσης, μπορεί να θεωρηθεί το event που προκαλείται από τη λήψη ενός πακέτου τύπου TOS_MsgPtr (βλέπε tos\types\AM.h). Το event αυτό, με τη σειρά του, καλεί την εντολή ProcessCmd.execute(pmsg) για την εκτέλεση της εντολής του σταθμού βάσης.

Β.2.2.3 command result_t ProcessCmd.execute(TOS_MsgPtr)

Η δομή του πεδίου `data` των πακέτων που λαμβάνονται καθορίζεται από το struct UARTInjectMsg. Γι'αυτό, στον αντίστοιχο ορισμό της ProcessCmd, η διάκριση της εντολής γίνεται από το πεδίο `action` του πακέτου, αφού προηγηθεί το κατάλληλο casting. Διακρίνονται οι εντολές START, STOP, ON, OFF και PROBE, οι οποίες ορίζονται στο UARTInject.h.

```
typedef struct UARTInjectMsg
{
      uint16_t action;
      uint16_t mpayload[3];
}uartMsg_t;
```

Β.2.2.4 async event result_t ADC.dataReady(uint16_t)

Το event αυτό προκαλείται επαναληπτικά από τον Timer του προγράμματος, με περίοδο που καθορίζεται σε runtime. Κάθε φορά που εκτελείται, επιστρέφεται μια τιμή από τον ADC και αποθηκεύεται σε πίνακα ADCBuf[TOSH_DATA_LENGTH].

Β.2.2.5 task void EEPROM_read();

Όταν ο πίνακας ADCBuf γεμίσει καλείται το task αυτό για να διαβάσει τα απαιτούμενα κλειδιά για την κρυπτογράφηση. Τα κλειδιά, τα οποία διαβάζονται (μια γραμμή 16 bytes κάθε φορά) με τη μέθοδο LoggerRead.read(rline,(uint8_t*)rlineBuf), αποθηκεύονται στον πίνακα rlineBuf. Στην περίπτωση που υπάρχουν, από προηγούμενη κλήση της read διαθέσιμα κλειδιά, καλείται άμεσα το task encrypt(), αλλιώς το πρόγραμμα μεταφέρεται στο event LoggerRead.readDone() από όπου καλείται το task encrypt().

B.2.2.6 task void encrypt();

Είναι το task που αναλαμβάνει την κρυπτογράφηση του πεδίου data κάθε μηνύματος. Σε περίπτωση που τα κλειδιά, τα οποία είναι φορτωμένα στη μνήμη δεν αρκούν καλείται εκ νέου η EEPROM_read(), διαφορετικά, το πρόγραμμα μεταβαίνει στο task forwardPacket().

B.2.2.7 task void forwardPacket()

Όταν ολοκληρωθεί η διαδικασία κρυπτογράφησης του μηνύματος, καλείται για την αποστολή του πακέτου προς τον σταθμό βάσης. Σε αυτό το task γίνεται διάκριση του τύπου μηνύματος σε AM_ADCMSG και AM_PROBEMSG για τα οποία ακολουθείται διαφορετική διαδικασία. Αφού συμπληρωθούν τα απαραίτητα για την αποστολή πεδία (length, key_pos, κ.λπ.), το πακέτο αποστέλεται με κλήση της RadioSend.send(&tx_packet);

```
/*
 * OTPTransceiverM.nc
 * University of Thessaly, 2006-2007
 * Arvanitis Dhionysis
 */

module OTPTransceiverM {
  provides {
    interface StdControl;
    interface ProcessCmd;
  }
  uses {
    interface Timer;
    interface Leds;
    interface ADC;

    interface StdControl as TimerControl;
    interface StdControl as ADCControl;
    interface StdControl as RadioControl;
    interface StdControl as LogControl;

    interface LoggerWrite;
    interface LoggerRead;

    interface BareSendMsg as RadioSend;
```

```
    interface ReceiveMsg as RadioReceive;

    interface OnOff;
    command result_t PowerEnable();
    command result_t PowerDisable();
  }
}

implementation {

  #include "client.h"
  #include "UARTInject.h"

  /* instance variables
  */
  TOS_Msg   tx_packet;
  TOS_Msg   rx_packet;

  uint8_t   buf_cnt,buf_iter,msg_cnt;
  uint16_t  key_cnt;
  uint16_t  total_packets;
  uint16_t  real_payload;
  uint16_t  msgBuf[TOSH_DATA_LENGTH],ADCBuf[TOSH_DATA_LENGTH];

  uint16_t  wlineBuf[TOS_EEPROM_LINE_SIZE/2];
  uint16_t  rlineBuf[TOS_EEPROM_LINE_SIZE/2];    //eeprom temp buffer
  uint16_t  keysInBuf;
  uint16_t  rline,wline;
  uint8_t   offset;          //eeprom offset (usually 16 lines)
  bool      eeprom_locked;   //eeprom flag
  bool      rf_locked;       //radio flag
  bool      probing;
  bool      sid_pending;
  bool      proc_flag;

  task void EEPROM_read();
  task void pad2flash();
  task void encrypt();
  task void forwardPacket();

  /*****************************************************************************
   * Initialization
   *****************************************************************************/

  command result_t StdControl.init()
  {
      result_t ok1,ok2,ok3,ok4,ok;
        atomic {
           buf_iter= buf_cnt= key_cnt= 0;
           rf_locked= FALSE;
           keysInBuf= 0;
           wline= rline= offset= 16;
           eeprom_locked= FALSE;
           probing= FALSE;
           sid_pending= TRUE;
           proc_flag= FALSE;
        }
        call OnOff.off();
      call PowerEnable();
      ok1= call TimerControl.init();
```

```
      ok2= call ADCControl.init();
      ok3= call RadioControl.init();
      ok4= call LogControl.init();

      if((ok= rcombine4(ok1,ok2,ok3,ok4)))
          dbg(DBG_BOOT, "MOTE.%d initialized\n",TOS_LOCAL_ADDRESS);

      return ok;
  }
  // start execution of the application.
  command result_t StdControl.start()
  {
      result_t ok1,ok2,ok3,ok4;
      ok1= call TimerControl.start();
      ok2= call ADCControl.start();
      ok3= call RadioControl.start();
      ok4= call LogControl.start();

      post pad2flash();
      //call Timer.start(TIMER_REPEAT, 100);
      return rcombine4(ok1,ok2,ok3,ok4);

  }
  // halt execution of the application.
  command result_t StdControl.stop()
  {
      result_t ok1,ok2,ok3;
      ok1 = call RadioControl.stop();
      ok2 = call ADCControl.stop();
      ok3 = call TimerControl.stop();
      return rcombine3(ok1,ok2,ok3);
  }
  /*****************************************************************************
  * Commands, events, tasks
  *****************************************************************************/
  /**
  * Recieved command handler.
  */
  command result_t ProcessCmd.execute(TOS_MsgPtr pmsg)
  {
    uartMsg_t* cmd= (uartMsg_t*)pmsg->data;
    switch (cmd->action){
      case START:
          atomic{
            total_packets= cmd->mpayload[0];
            real_payload= cmd->mpayload[1];
            call Timer.start(TIMER_REPEAT,cmd->mpayload[2]);
          }
          break;
      case STOP:
            call Timer.stop();
            dbg(DBG_USR1,"MOTE stop\n");
          break;
      case ON :
          if( cmd->mpayload[0]== TOS_LOCAL_ADDRESS || cmd->mpayload[0]==
(uint8_t)0xffff ){
              call OnOff.on();
              dbg(DBG_USR1,"MOTE woke up\n");
          }
          break;
```

```
        case OFF:
             if( cmd->mpayload[0]== TOS_LOCAL_ADDRESS || cmd->mpayload[0]==
(uint8_t)0xffff ){
                  call OnOff.off();
                  dbg(DBG_USR1,"MOTE sleep\n");
             }
             break;
        case PROBE:
           atomic{
                  tx_packet= *pmsg;
                  tx_packet.type= AM_PROBEMSG;
                  probing= TRUE;
                  post EEPROM_read();
             }
    }
    return SUCCESS;
  }
  /**
  * Write pad to eeprom task.
  */
  task void pad2flash()
  {
    atomic {
        uint8_t* ptr; uint8_t in;
        if(buf_iter< PAD_SIZE)
        {
            for(in=0; in<(TOS_EEPROM_LINE_SIZE/2); in++)
                wlineBuf[in]= pad[buf_iter++];
            ptr= (uint8_t*)wlineBuf;
            call LoggerWrite.write(wline,ptr);
            if(TOS_LOCAL_ADDRESS!=0)
            wline++;
        }
    }
  }
  /**
  * Event handler to the Timer.fired event.
  */
  event result_t Timer.fired()
  {
    if (total_packets != 0 && !proc_flag) {
        call ADC.getData();
    } else {
        call Leds.redToggle();
    }
    return SUCCESS;
  }
  /**
  * Event handler to the ADC.dataReady() event.
  */
  async event result_t ADC.dataReady(uint16_t _data)
  {
      uint16_t temp=_data;
      atomic {
           ADCBuf[buf_cnt]= temp;
           if( ++buf_cnt== real_payload ){
               dbgMessage(ADCBuf,real_payload,"ADC");
               buf_cnt= 0; proc_flag=TRUE;
               post EEPROM_read();
           }
```

```
        }
        return SUCCESS;
    }
    /**
    * Read pad-keys from eeprom.
    */
    task void EEPROM_read()
    {
        if( rline< ( PAD_SIZE/(TOS_EEPROM_LINE_SIZE/2)+offset) &&
!eeprom_locked ){
            if(!keysInBuf) {
                call LoggerRead.read(rline,(uint8_t*)rlineBuf);
                eeprom_locked= TRUE;
                keysInBuf= (TOS_EEPROM_LINE_SIZE/2);
            }
            else { post encrypt(); }
        }
        else if(!eeprom_locked){
            if(keysInBuf){post encrypt();}else{
            dbg(DBG_USR1,"Not enough keys. Reseting pad...\n");
            rline=offset;
            eeprom_locked = TRUE;
            call LoggerRead.read(rline,(uint8_t*)rlineBuf);
            keysInBuf= (TOS_EEPROM_LINE_SIZE/2);}
        }
        else{ dbg(DBG_USR1,"eeprom pending\n"); }
    }
    /**
    * Encryprion task.
    */
    task void encrypt()
    {
    uint8_t i;
    atomic {
        if(sid_pending){
            packet_t* m= (packet_t*)tx_packet.data;
            m->source_id=
TOS_LOCAL_ADDRESS^rlineBuf[key_cnt%(TOS_EEPROM_LINE_SIZE/2)];
            key_cnt++; keysInBuf--;
            sid_pending= FALSE;
        }
        for(i=0;i<(TOS_EEPROM_LINE_SIZE/2);i++){
            if(keysInBuf== 0) {
                post EEPROM_read();
                break;
            }
            msgBuf[msg_cnt]=
ADCBuf[msg_cnt]^rlineBuf[key_cnt%(TOS_EEPROM_LINE_SIZE/2)];
            key_cnt++; keysInBuf--;
            if(++msg_cnt== real_payload) {
                tx_packet.type= AM_ADCMSG;
                proc_flag= FALSE;
                post forwardPacket();
                break;
            }
            else if(keysInBuf== 0) {
              post EEPROM_read();
              break;
            }
        }
```

```
  }}
  /**
  * Data transmission.
  */
  task void forwardPacket()
  {
      packet_t* m= (packet_t*)tx_packet.data;
      if(!rf_locked) {
          rf_locked= TRUE;
          atomic {
              tx_packet.addr= TOS_BS_ADDR;
              tx_packet.group= (TOS_AM_GROUP & 0xff);
              switch(tx_packet.type){
                case AM_ADCMSG :
                    sid_pending= TRUE;
                    m->key_pos= (key_cnt-real_payload-1)%PAD_SIZE;
                     tx_packet.length= real_payload*2+4;
                    memcpy(m->payload,msgBuf,real_payload*2);
                    total_packets--; msg_cnt= 0;
                    dbgMessage(msgBuf,real_payload,"ENC");
                    break;
                case AM_PROBEMSG :
                    m->source_id=
TOS_LOCAL_ADDRESS^rlineBuf[key_cnt%(TOS_EEPROM_LINE_SIZE/2)];
                    m->key_pos= key_cnt;
                    key_cnt++; keysInBuf--;
                    m->payload[0]=
NL_KEY^rlineBuf[key_cnt%(TOS_EEPROM_LINE_SIZE/2)]; //NL-key encryption
                    key_cnt++; keysInBuf--;
                    tx_packet.length= 24;
                    probing= FALSE;
                    break;
            }
            dbg(DBG_USR2,"mote.%d sends packet to
radio.\n",TOS_LOCAL_ADDRESS);
            dbgPacket(&tx_packet,"Sent");
            call RadioSend.send(&tx_packet);
        }
     } else {
         dbg(DBG_USR1,"mote.%d transmission failure.\n",TOS_LOCAL_ADDRESS);
     }
  }
 /**
 * Checks AM type and calls ProcessCmd.execute().
 */
  event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr pmsg)
  {
        if(pmsg->addr!= TOS_BS_ADDR){
            dbg(DBG_USR2,"mote.%d received radio
packet.\n",TOS_LOCAL_ADDRESS);
            dbgPacket(pmsg,"Received");
            call ProcessCmd.execute(pmsg);
        }
        return pmsg;
  }
  // indicate the completion of transfer.
  event result_t RadioSend.sendDone(TOS_MsgPtr _msg, result_t status)
  {
      if(_msg == &tx_packet && status){
          rf_locked = FALSE;
```

```
            if(total_packets== 0) {call Timer.stop();}
            return SUCCESS;}
        else { return FAIL; } //post forwardPacket();
    }
    // indicate the completion of eeprom write.
    event result_t LoggerWrite.writeDone(result_t status)
    {
        if(!status){
            wline--;
            buf_iter-=(TOS_EEPROM_LINE_SIZE/2);
            call LoggerWrite.setPointer(wline);
            dbg(DBG_USR1,"Logger.Write.Error at line: %d\n",wline);
        }
        post pad2flash();
        return SUCCESS;
    }
    // If eeprom read succeed calls encrypt method. Try again otherwise.
    event result_t LoggerRead.readDone(uint8_t* _buffer, result_t status)
    {
        if(_buffer==(uint8_t*)rlineBuf){
            if(probing==FALSE){
               post encrypt();
            }else {
                 post forwardPacket();
            }
            rline++;
        }else {
            if(TOS_LOCAL_ADDRESS!=0)
                dbg(DBG_USR1,"Logger.Read.Error at line: %d\n",rline);
                post EEPROM_read();
        }
        atomic eeprom_locked = FALSE;
        return status;
    }
    // Sets mote to low power listening mode
    event result_t OnOff.requestOff()
    {
        call PowerEnable();
        return SUCCESS;
    }
    // Sets mote to active power listening mode
    event result_t OnOff.ison()
    {
        call PowerDisable();
        return SUCCESS;
    }

}
```

Β.3.1 TOSBase.nc

Για το configuration αρχείο του σταθμού βάσης αξίζει να παρατηρηθεί τα παραμετροποιημένα interface των εντολών `SendMsg` και `ReceiveMsg` της UART. Με αυτόν τον τρόπο, δηλώνεται στον compiler ότι η αποστολή και λήψη μέσω σειριακής, γίνεται για συγκεκριμένο τύπο μηνύματος. Για παράδειγμα, με το wiring

```
TOSBaseM.UARTReceive -> UART.ReceiveMsg[AM_UARTINJECTMSG];
```

η UARTReceive του TOSBaseM, ακούει αποκλειστικά για πακέτα τύπου AM_UARTINJECTMSG.

```
/*
 * TOSBase.nc
 * University of Thessaly, 2006-2007
 * Arvanitis Dhionysis
 */

includes UARTInject;
includes OscopeMsg;

configuration TOSBase {
  provides interface StdControl;
}
implementation {
  components TOSBaseM,
             LedsC,
             RadioCRCPacket as COM,
             UARTComm as UART,
             Logger;

  StdControl = TOSBaseM.StdControl;

  TOSBaseM.RadioControl -> COM.Control;
  TOSBaseM.RadioSend -> COM;
  TOSBaseM.RadioReceive -> COM;

  TOSBaseM.UARTControl -> UART.Control;
  TOSBaseM.UARTSend -> UART.SendMsg[AM_OSCOPEMSG];
  TOSBaseM.UARTReceive -> UART.ReceiveMsg[AM_UARTINJECTMSG];
  TOSBaseM.ResetCounterMsg -> UART.ReceiveMsg[AM_OSCOPERESETMSG];

  TOSBaseM.LogControl -> Logger.StdControl;
  TOSBaseM.LoggerWrite -> Logger.LoggerWrite;
  TOSBaseM.LoggerRead -> Logger.LoggerRead;

  TOSBaseM.Leds -> LedsC;
}
```

Β.3.2 TOSBaseM.nc

Ακολουθεί η περιγραφή των κυριότερων μεθόδων του module, το οποίο υλοποιεί τη λειτουργικότητα του σταθμού βάσης.

Β.3.2.1 event TOS_MsgPtr UARTReceive.receive(TOS_MsgPtr)

Το event αυτό προκαλείται από τη λήψη ενός πακέτου μέσω σειριακής. Τα πακέτα αυτά είναι εντολές προς τους κόμβους του δικτύου γι' αυτό και προωθούνται απευθείας για αποστολή με κλήση της συνάρτησης forward2Radio(TOS_MsgPtr).

Β.3.2.2 void forward2Radio(TOS_MsgPtr)

Πριν την αποστολή του πακέτου προς τους κόμβους του δικτύου, κάθε πακέτο αποθηκεύεται σε κυκλική ουρά TxQ[TX_QUEUE_LEN] και ακολούθως αν εκείνη την ώρα δεν εκκρεμεί κάποια αποστολή (txBusy) γίνεται post το RadioSendTask().

Β.3.2.3 task void RadioSendTask()

Με διαδοχικές αναδρομικές κλήσεις μέσω του event RadioSend.sendDone() αποστέλονται όλα τα πακέτα που εκκρεμούν στην κυκλική ουρά, αφού προηγούμενως καθοριστεί το groupID και η broadcast διεύθυνση κάθε πακέτου.

Β.3.2.4 event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr)

Κάθε λήψη μηνύματος από τους κόμβους προκαλεί αυτό το event. Το πεδίο data του μηνύματος έχει τη μορφή της παρακάτω δομής

```
typedef struct packet {
      uint16_t source_id;
      uint16_t key_pos;
      uint16_t payload[10];
}packet_t;
```

Αρχικά, γίνεται φιλτράρισμα του πακέτου που λαμβάνεται με έλεγχο των CRC, groupID και διεύθυνσης παραλήπτη. Σε περίπτωση που το πακέτο δεν απορριφθεί, γίνεται διάκριση τύπου μηνύματος σε AM_ADCMSG για μηνύματα που περιέχουν τιμές ADC και AM_PROBEMSG για μηνύματα που απαντούν στην εντολή PROBE για αυθεντικοποίηση.

Στην πρώτη περίπτωση, το μήνυμα τοποθετείται σε κυκλική ουρά uartQ[UART_QUEUE_LEN] αφού προηγουμένως γίνει έλεγχος διαθέσιμου χώρου και καλείται το task processADC(). Στην

περίπτωση AM_PROBEMSG, εντοπίζεται η γραμμή (rline) από την οποία πρέπει να ξεκινήσει να διαβάζει ο σταθμός βάσης για την κρυπτογράφηση του πακέτου και ακολούθως εκτελείται ανάγνωση των κλειδιών από τη μνήμη EEPROM.

B.3.2.5 task void processADC()

Τα πακέτα τύπου AM_ADCMSG που περιέχουν τις τιμές του ADC, επεξεργάζονται από αυτό το task. Συγκεκριμένα, διαβάζεται η γραμμή και η θέση του κλειδιού από το οποίο πρέπει να ξεκινήσει η αποκρυπτογράφηση του πακέτου και στη συνέχεια διαβάζονται τα κλειδιά από την EEPROM.

B.3.2.6 task void decrypt()

Όταν ολοκληρωθεί η ανάγνωση των κλειδιών από τη σταθερή μνήμη με κλήση του task EEPROMread(), εκτελείται η αποκρυπτογράφηση του μηνύματος με την εκτέλεση της παρακάτω πράξης εντός βρόχου for (έχει προηγηθεί #define xor ^):

```
m->payload[msg_cnt]= m->payload[msg_cnt]^rlineBuffer[key_line_idx];
```

Για την αποκρυπτογράφηση των τιμών χρησιμοποιείται η ίδια δομή του πακέτου payload για εξοικονόμηση χώρου και χρόνου. Η ολοκλήρωση της διαδικασίας καθορίζεται από το πεδίο length του πακέτου. Στη συνέχεια, δημιουργείται μήνυμα τύπου OscopeMsg και αντιγράφονται σε αυτό οι τιμές του ADC και συμπληρώνονται τα υπόλοιπα απαραίτητα πεδιά του. Το τελικό μήνυμα προωθείται στην UART για να προβληθούν οι αποκρυπτογραφημένες τιμές στην εφαρμογή Oscilloscope.

```
struct OscopeMsg
{
    uint16_t sourceMoteID;
    uint16_t lastSampleNumber;
    uint16_t channel;
    uint16_t data[BUFFER_SIZE];
};
```

B.3.2.7 task void UARTSendTask()

Αφού καθοριστεί κατάλληλα το πακέτο, αποστέλεται στη διεύθυνση της UART (0x007e) με κλήση της παρακάτω εντολής:

```
call UARTSend.send(TOS_UART_ADDR,sizeof(struct OscopeMsg),&uartQ[uartOut]);
```

```
/*
 * TOSBaseM.nc
 * University of Thessaly, 2006-2007
 * Arvanitis Dhionysis
 */

includes OscopeMsg;
module TOSBaseM {
  provides interface StdControl;
  uses {

    interface StdControl as RadioControl;
    interface BareSendMsg as RadioSend;
    interface ReceiveMsg as RadioReceive;

    interface StdControl as UARTControl;
    interface SendMsg as UARTSend;
    interface ReceiveMsg as UARTReceive;
    interface ReceiveMsg as ResetCounterMsg;

    interface StdControl as LogControl;

    interface LoggerWrite;
    interface LoggerRead;

    interface Leds;
  }
}

implementation
{
  #include "base.h"
  #include "UARTInject.h"

  TOS_Msg         tx_packet;
  TOS_Msg         proc_msg;

  TOS_Msg         TxQ[TX_QUEUE_LEN];  // circular buffer
  uint8_t         txIn, txOut;
  bool            txBusy, txCount;

  TOS_Msg         uartQ[UART_QUEUE_LEN];  // circular buffer
  uint8_t         uartIn, uartOut;
  bool            uartBusy, uartCount;

  TOS_MsgPtr      rmsg_copy;
  uint16_t        nodesList[TOTAL_NODES];
  uint8_t         nodelist_i;

  task void       UARTSendTask();
  task void       RadioSendTask();

  TOS_MsgPtr      probeMsg;
  bool            probing;
  bool            auth_pending,auth_ok;
  task void probe();

  uint16_t  wlineBuffer[TOS_EEPROM_LINE_SIZE/2];
  uint16_t  rlineBuffer[TOS_EEPROM_LINE_SIZE/2];
```

```
  uint8_t   key_line_idx,buf_iter,msg_cnt;
  uint16_t  rline,wline;
  uint8_t   offset;
  bool      eeprom_locked;
  uint16_t  lsnum;

  task void EEPROMread();
  task void pad2flash();
  task void forward2UART();
  task void decrypt();
  task void processADC();

  bool authenticate(uint16_t);
  bool checkValidity(uint16_t);

  void failBlink();
  void dropBlink();
  void forward2Radio(TOS_MsgPtr);

  /*****************************************************************************
   * Initialization
   *****************************************************************************/
  // initialise execution of the application.
  command result_t StdControl.init()
  {
    result_t ok1,ok2,ok3,ok4,ok;

    buf_iter= 0;
    wline= rline= offset= 16;
    eeprom_locked= FALSE;
    probing= FALSE;
    auth_pending= TRUE;
    lsnum=0;
    txIn = txOut = txCount = 0;
    txBusy = FALSE; uartBusy = FALSE;
    uartIn = uartOut = uartCount = 0;
    nodelist_i =0; rmsg_copy=NULL;

    ok1 = call UARTControl.init();
    ok2 = call RadioControl.init();
    ok3 = call LogControl.init();
    ok4 = call Leds.init();
    call LoggerWrite.resetPointer();
    if( (ok=rcombine4(ok1,ok2,ok3,ok4)) )
      dbg(DBG_BOOT, "BS initialized\n");
    return ok;
  }
  // start execution of the application.
  command result_t StdControl.start()
  {
    result_t ok1,ok2;
    ok1 = call UARTControl.start();
    ok2 = call RadioControl.start();
    call LogControl.start();

    post pad2flash();
    return rcombine(ok1,ok2);
  }
  // halt execution of the application.
  command result_t StdControl.stop()
```

```
{
  result_t ok1,ok2;
  ok1 = call UARTControl.stop();
  ok2 = call RadioControl.stop();
  return rcombine(ok1,ok2);
}
// creates packet to find the valid nodes of network
task void probe()
{
    uartMsg_t* m = (uartMsg_t*)tx_packet.data;
    tx_packet.length=2;
    m->action=PROBE;
    forward2Radio(&tx_packet);
}
// adds to nodelist the valid nodes
void addnode(TOS_MsgPtr Msg)
{
    packet_t* m = (packet_t*)Msg->data;
    uint8_t node_id= m->source_id^rlineBuffer[m->key_pos];
    uint16_t NLK_auth= m->payload[0]^rlineBuffer[m->key_pos+1];
    if( checkValidity(node_id) && NLK_auth==NL_KEY){
        if(TOS_LOCAL_ADDRESS==0)
           dbg(DBG_USR1, "Node.%d authentication succeed\n",node_id);
        nodesList[nodelist_i] = node_id;
    }
    nodelist_i++;
    probing= FALSE;
}
/**
* Node validation checking
*/
bool checkValidity(uint16_t sid){
   uint8_t i;
   for(i=0;i<TOTAL_NODES;i++){
       if( validNodes[i]== sid)
          return TRUE;
    }
    return FALSE;
}
/**
* Node authentication checking.
*/
bool authenticate(uint16_t sid){
   uint8_t i;
   for(i=0;i<TOTAL_NODES;i++){
       if( nodesList[i]== sid)
          return TRUE;
    }
    return FALSE;
}
/**
* Write pad to eeprom task.
*/
task void pad2flash()
{
  atomic {
      uint8_t* ptr; uint8_t in;
      if(buf_iter<PAD_SIZE){
          for(in=0;in<(TOS_EEPROM_LINE_SIZE/2);in++)
              wlineBuffer[in] = pad[buf_iter++];
```

```
            ptr=(uint8_t*)wlineBuffer;
            call LoggerWrite.write(wline,ptr);
            wline++;
        }else{ post probe();}
    }
  }
  /**
  * Read pad-keys from eeprom.
  */
  task void EEPROMread()
  {
      if( rline<(PAD_SIZE/(TOS_EEPROM_LINE_SIZE/2)+offset)&&!eeprom_locked)
      {
          call LoggerRead.read(rline,(uint8_t*)rlineBuffer);
      }
      else if(!eeprom_locked){
          dbg(DBG_USR1,"Not enough keys. Reseting pad...\n");
          rline= offset; key_line_idx= 0;
          call LoggerRead.read(rline,(uint8_t*)rlineBuffer);
      }
      else{ dbg(DBG_USR1,"eeprom pending\n"); }
      eeprom_locked= TRUE;
  }
  /**
  * Message decryption.
  */
  task void decrypt()
  {
    uint8_t i;
    atomic {
        packet_t* m= (packet_t*)proc_msg.data;

        for(i= key_line_idx; i<TOS_EEPROM_LINE_SIZE/2; i++) {
            m->payload[msg_cnt]= m-
>payload[msg_cnt]^rlineBuffer[key_line_idx];
            if(TOS_LOCAL_ADDRESS== 0){
            dbg(DBG_USR1, "Decrypt mote.%d> ADC:%d with KEY:%d\n",m-
>source_id, m->payload[msg_cnt],rlineBuffer[key_line_idx]);
            }
            key_line_idx++;
            if(++msg_cnt== proc_msg.length/2-2) {
                  /* create message for UART */
                  TOS_Msg uart_msg;
                  struct OscopeMsg* ureply= (struct
OscopeMsg*)uart_msg.data;
                uint8_t ii;
                key_line_idx= 0;
              uart_msg.type= AM_OSCOPEMSG;

              for(ii=0; ii<proc_msg.length/2-2; ii++){
                  ureply->data[ii]= m->payload[ii];
              }

              ureply->lastSampleNumber= lsnum;
              ureply->channel= 0; //or m->source_id-1 for 2 channels;
              ureply->sourceMoteID= m->source_id;

              uartQ[uartOut]= uart_msg;
              uartBusy = TRUE;
                post UARTSendTask();
```

```
                break;
            }
        }
        if(msg_cnt!= proc_msg.length/2-2) {
            key_line_idx= 0;
            post EEPROMread(); //get more keys if necessary
        }
    }
  }
  task void processADC()
  {
      packet_t* m= (packet_t*)uartQ[uartOut].data;
      rline= m->key_pos/8+ offset;
      key_line_idx= m->key_pos% 8;
      proc_msg= uartQ[uartOut];
      msg_cnt= 0;
      post EEPROMread();
  }

  event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr pmsg)
  {
    packet_t* m;
    //packet filtering for CRC, groupID and destination address.
    if ((!pmsg->crc) || (pmsg->group != TOS_AM_GROUP) || (pmsg-
>addr!=TOS_LOCAL_ADDRESS)){
      if(TOS_LOCAL_ADDRESS== 0){
          dbg(DBG_USR1, "BS rejects packet.\n");
        }
        return pmsg;
    }
    dbgPacket(pmsg,"Received");
    if (uartCount < UART_QUEUE_LEN)
    {
        switch(pmsg->type) {
            case AM_ADCMSG :
                memcpy(&uartQ[uartIn], pmsg, sizeof(TOS_Msg));
                uartCount++; auth_pending= TRUE;
                if( ++uartIn >= UART_QUEUE_LEN ) uartIn = 0;
                post processADC();
                break;
            case AM_PROBEMSG :
                dbg (DBG_USR1, "AM_PROBEMSG received\n");
                probing= TRUE;
                probeMsg= pmsg;
                m= (packet_t*)pmsg->data;
                rline= m->key_pos/8+ offset;
                post EEPROMread();
      }
    }
    return pmsg;
  }

  task void UARTSendTask()
  {
      if (uartCount == 0)
          uartBusy = FALSE;
      else {
          uartQ[uartOut].length=sizeof(struct OscopeMsg);
          lsnum+=uartQ[uartOut].length/2;
          call UARTSend.send(TOS_UART_ADDR,sizeof(struct
```

```
OscopeMsg),&uartQ[uartOut]);
          dbgPacket(&uartQ[uartOut],"Sent");
      }
  }

  event TOS_MsgPtr UARTReceive.receive(TOS_MsgPtr Msg)
  {
    forward2Radio(Msg);
    return Msg;
  }

  void forward2Radio(TOS_MsgPtr Msg)
  {
    if(TOS_LOCAL_ADDRESS==0)
      dbg(DBG_USR2, "BS forwarding packet to Radio\n");
    if (txCount < TX_QUEUE_LEN) {
      memcpy(&TxQ[txIn], Msg, sizeof(TOS_Msg));
      txCount++;
      if( ++txIn >= TX_QUEUE_LEN ) txIn = 0;
      atomic {
            if (!txBusy) {
            rmsg_copy=&TxQ[txOut];  //for event done checking
            post RadioSendTask();
            txBusy = TRUE;
            }
      }
    }else { dropBlink(); dbg(DBG_USR1, "BS packet drop\n"); }
  }

  task void RadioSendTask()
  {
    if (txCount == 0) {
      txBusy = FALSE;
    }
    else {
        TxQ[txOut].group = TOS_AM_GROUP;
        TxQ[txOut].addr = TOS_BCAST_ADDR;
        if (call RadioSend.send(&TxQ[txOut]) == SUCCESS) {
            dbg(DBG_USR1, "BS sends packet to radio\n");
            dbgPacket(&TxQ[txOut],"Sent");
          call Leds.redToggle();
      }
      else { failBlink(); post RadioSendTask(); }
    }
  }

  event result_t RadioSend.sendDone(TOS_MsgPtr msg_, result_t success)
  {
    if( msg_!=rmsg_copy || !success ){
        failBlink();
    }
    else {
      txCount--;
      if( ++txOut >= TX_QUEUE_LEN ) txOut = 0;
    }
    post RadioSendTask();
    return SUCCESS;
  }

  event result_t UARTSend.sendDone(TOS_MsgPtr msg_, result_t success)
```

```
  {
    if(!success) {
       failBlink();
    } else {
      auth_pending= TRUE;
      uartCount--;
      if( ++uartOut >= UART_QUEUE_LEN ) uartOut = 0;
    }
    return SUCCESS;
  }

  event result_t LoggerWrite.writeDone(result_t status)
  {
      if(!status){
          wline--;
          buf_iter-=(TOS_EEPROM_LINE_SIZE/2);
          call LoggerWrite.setPointer(wline);
      }
      post pad2flash();
      return SUCCESS;
  }

  event result_t LoggerRead.readDone(uint8_t* _buffer, result_t status)
  {
      if(probing== FALSE){
         if(auth_pending){
            packet_t* m= (packet_t*)proc_msg.data;

            m->source_id= m->source_id^rlineBuffer[key_line_idx];
            auth_ok= authenticate(m->source_id);
            key_line_idx++;
            auth_pending= FALSE;
        }
        if(auth_ok== TRUE){
          post decrypt();
          rline++;
        }
      }
      else{ addnode(probeMsg); }
      atomic eeprom_locked = FALSE;
      return status;
  }

  event TOS_MsgPtr ResetCounterMsg.receive(TOS_MsgPtr m) {
    atomic lsnum= 0;
    return m;
  }

  void dropBlink() {
#ifdef TOSBASE_BLINK_ON_DROP
    call Leds.yellowToggle();
#endif
  }

  void failBlink() {
#ifdef TOSBASE_BLINK_ON_FAIL
    call Leds.yellowToggle();
#endif
  }
}
```

B.4.1 ProcessCmd.nc

Το αρχείο αυτό περιλαμβάνει το interface για τις εντολές που δίνονται από το σταθμό βάσης προς τους κόμβους του δικτύου. Η μέθοδος execute εκτελεί την εντολή και το event done δηλώνει την ολοκλήρωση της διαδικασίας.

```
includes AM;
/**
 * This interface process a
 * command and is capable of the handling of available commands
 */
interface ProcessCmd
{
  /**
   * Extracts the command from the message 'pmsg' and
   * executes the command.
   */
  command result_t execute(TOS_MsgPtr pmsg);

  /**
   * Indicate that the command contained in 'pmsg' has finished executing.
   */
  event result_t done(TOS_MsgPtr pmsg, result_t status);
}
```

B.4.2 client.h

Έχει δημιουργηθεί για να περιλαμβάνει τιμές που αφορούν μόνο τους clients του δικτύου. Για τη συγκεκριμένη προσομοίωση χρειάζεται να γνωρίζουν μόνο τη διεύθυνση του σταθμού βάσης (0x00)

```
#ifndef _CLIENT_H
#define _CLIENT_H

#include "common_types.h"
#define TOS_BS_ADDR 0x00

#endif
```

B.4.3 base.h

Εδώ μπορεί να καθοριστεί το μέγεθος των δύο κυκλικών ουρών TxQ, UartQ του σταθμού βάσης. Το πεδίο TOTAL_NODES αφορά το πλήθος των κόμβων-πελατών της προσομοίωσης.

```
#ifndef _BASE_H
#define _BASE_H

#include "common_types.h"

#define  TOSBASE_BLINK_ON_DROP
#define  TOSBASE_BLINK_ON_FAIL
#define  TX_QUEUE_LEN          12
#define  UART_QUEUE_LEN        12
#define  TOTAL_NODES           2
```

```
uint8_t validNodes[] = {1,2}; //for authentication

#endif
```

B.4.4 common_types.h

Το common_types.h γίνεται include από τον σταθμό βάσης και από τους κόμβους του δικτύου. Περιλαμβάνει το format του πακέτου που στέλνουν οι κόμβοι του δικτύου στον σταθμό βάσης και τα κλειδιά της pad που πρόκειται να γραφούν στην EEPROM κατά την εκκίνηση των κόμβων. Τα AM types AM_PROBEMSG και AM_ADCMSG, τα οποία αποστέλλονται με κάθε πακέτο, καθορίζουν αν πρόκειται για μήνυμα αυθεντικοποίησης ή για μήνυμα συλλογής δεδομένων, αντίστοιχα.

Τέλος, η debugging συνάρτηση dbgPacket, τυπώνει στην οθόνη πακέτα τύπου TOS_MsgPtr, ενώ η συνάρτηση dbgMessage χρησιμοποιείται για την εκτύπωση πινάκων, όπως το περιεχόμενο του ADCBuf.

```
#ifndef _COMMON_TYPES_H
#define _COMMON_TYPES_H
#include <AM.h>

enum {
    PAD_SIZE = 120, //must be x8, PAD_SIZE < TOS_EEPROM_MAX_LINES(32768)
    AM_PROBEMSG = 6,
    AM_ADCMSG = 3,
};

#define xor ^
#define NL_KEY 0x59d8

/* data packet format */
typedef struct packet {
      uint16_t source_id;
      uint16_t key_pos;
      uint16_t payload[10];
}packet_t;

uint16_t pad[] = {
  0xc791,  0xa6a, 0x2fb9, 0xeac2, 0x2ae6, 0x2357, 0x81e3, 0x807e,
  0x9534, 0xf089, 0x2117, 0xd6e5, 0xcb69, 0xda4e, 0x5f59, 0xc5a6,
  0xce7c, 0x9d11, 0xe9b4, 0xced6, 0x1a6d, 0xb19f, 0xb5f,  0x391,
  0x1a28, 0xf5aa, 0x2cca, 0xfedc, 0x9d7e, 0x1712, 0xc8b6, 0x5d02,
  0x33b8, 0x3a75, 0x3956, 0xf985, 0x73f0, 0x2a85, 0xf58e, 0x3d3,
  0x1fda, 0xe999, 0xeb75, 0x2023, 0xb8b8, 0xbdd,  0x9a68, 0xd718,
  0xaf41, 0xa1ad, 0x9322, 0x9ef4, 0xb1c4, 0xbc56, 0x3005, 0x4b7f,
  0xaefc, 0xd28e, 0xbead, 0x59d8, 0xc674, 0xdda3, 0x5519, 0x21bc,
  0x176c, 0xa765, 0xfa6f, 0x98da, 0xc91e, 0x71ad, 0xcb8d, 0xed0a,
  0x8ac6, 0x65f3, 0x71f4, 0x47fc, 0x4d75, 0xe860, 0x7a04, 0x6090,
  0xb3db, 0x2b2b, 0x50dd, 0x1201, 0xc7bc, 0xd76b, 0xf6cd, 0x5f20,
  0x4009, 0x75c5, 0x1e65, 0x9052, 0x3c9,  0xd0d6, 0x4dfb, 0x4a1b,
  0x64da, 0x2cd,  0x29c2, 0xa1bf, 0x3a75, 0xb5f,  0x33b8, 0x5f20,
  0xa6a,  0xfedc, 0x2a85, 0xcb8d, 0x1fda, 0x50dd, 0xb5f, 0xc5a6,
  0xb3db, 0x2b2b, 0x50dd, 0x1201, 0xc7bc, 0xd76b, 0xf6cd, 0x5f20
```

```
};

void dbgPacket(TOS_MsgPtr data, char* cstr) {
      uint8_t i;
      dbg(DBG_USR1,"%s message:\n",cstr);
      dbg_clear(DBG_USR1, "\t");
      for(i=0; i<sizeof(TOS_Msg); i++) {
            dbg_clear(DBG_USR1, "%02hhx ", ((uint8_t *)data)[i]);
      }
      dbg_clear(DBG_USR1, "\n");
}

void dbgMessage(uint16_t* data, uint8_t length, char* cstr) {
      uint8_t i;
      dbg(DBG_USR1,"%s data:\n",cstr);
      dbg_clear(DBG_USR1, "\t");
      for(i=0; i<length; i++) {
            dbg_clear(DBG_USR1, "%d ", data[i]);
      }
      dbg_clear(DBG_USR1, "\n");
}
#endif
```

B.4.5 OscopeMsg.h

Το header αυτό γίνεται include από το σταθμό βάσης. Περιλαμβάνει τη μορφή του πακέτου και τα AM types που αφορούν την εφαρμογή Oscilloscope. Το κενό struct OscopeResetMsg είναι το πακέτο που αποστέλεται στον σταθμό βάσης από το Oscilloscope για την επανεκκίνηση της εφαρμογής. Κατά τη λήψη ενός τέτοιου μηνύματος μηδενίζεται η τιμή που καθορίζει το lastSampleNumber (μεταβλητή lsnum στο TOSBaseM.nc).

```
#ifndef _OSCOPEMSG_H
#define _OSCOPEMSG_H

enum {
    BUFFER_SIZE = 10
    AM_OSCOPEMSG = 10,
    AM_OSCOPERESETMSG = 32
};

struct OscopeMsg
{
    uint16_t sourceMoteID;
    uint16_t lastSampleNumber;
    uint16_t channel;
    uint16_t data[BUFFER_SIZE];
};

struct OscopeResetMsg
{
    /* Empty payload!*/
};

#endif
```

B.4.6 UARTInject.h

Στο header αυτό γίνεται ο ορισμός του τύπου των εντολών, καθώς και το active message (AM) type του μηνύματος inject AM_UARTINJECTMSG. Η δομή UARTInjectMsg, η οποία αποτελείται από το πεδίο action (τύπος εντολής) μεγέθους 2 bytes και τον πίνακα mpayload μέγιστου μεγέθους 3 τιμών, 2 bytes η καθεμία για να μεταφέρει τις παραμέτρους της εντολής START.

```
#ifndef _UART_PACKET_H
#define _UART_PACKET_H

enum {
  START = 2,
  STOP = 3,
  ON = 4,
  OFF = 5,
  PROBE = 6,
  AM_UARTINJECTMSG = 1
};

typedef struct UARTInjectMsg
{
      uint16_t action;
      uint16_t mpayload[3];
}uartMsg_t;

#endif
```

B.5 Inject.java

Το αρχείο αυτό υλοποιεί τη λειτουργικότητα του packet injection στο δίκτυο μέσω σειριακής UART.

Με τη βοήθεια του εργαλείου MIG (Message Interface Generator), το οποίο περιλαμβάνεται στο TinyOS, παράγεται αρχείο class για την κωδικοποίηση και αποκωδικοποίηση των πακέτων. Η παρακάτω εντολή

```
#!/bin/sh

mig java -java-classname=UARTInjectMsg UARTInject.h UARTInjectMsg -
oUARTInjectMsg.java
```

παίρνει το struct του μηνύματος από το header UARTInject.h, το οποίο ανταλλάσεται μεταξύ UART και σταθμού βάσης και δημιουργεί την κλάση

```
public class UARTInjectMsg extends net.tinyos.message.Message
```

Το αρχείο Inject.java δημιουργεί αντικείμενο της κλάσης MoteIF με τη βοήθεια της οποίας γίνεται η σύνδεση με τον SerialForwarder του TinyOS για την αποστολή των πακέτων.

```
/**
 * Send one or more a messages to the UART.
 */

import net.tinyos.message.*;
import net.tinyos.util.*;

import java.io.*;
import java.text.*;

public class Inject implements MessageListener
{
  static final short TOS_UART_ADDR = 0x007e;
  private String m_strings[];
  private MoteIF m_moteif;

  public static void usage() {
      System.err.println("Usage: java Inject"+
                         " <command> [arguments]");
      System.err.println("\t\tprobe");
      System.err.println("\t\tstart [npackets nsamples interval_ms]");
      System.err.println("\t\tstop ");
      System.err.println("\t\ton");
      System.err.println("\t\toff");
      System.exit(-1);
  }

  Inject( String[] args )
  {
    if( args.length <= 0 ) {
        usage();
    }

    try
    {
      m_moteif = new MoteIF((Messenger)null);
    }
    catch (Exception e)
    {
      System.out.println("ERROR: Couldn't contact serial forwarder.");
      System.exit(1);
    }
    m_strings = args;
    m_moteif.start();
  }

  public synchronized void send( String[] str )
  {
    try
    {
        UARTInjectMsg m = new UARTInjectMsg();
        int[] vals = new int[str.length-1];
        int action= 0;
      if( str[0].equals("start") ) {
```

```
            action=2;
          System.out.print( "Send> ");
          System.out.print("start"+" ");
          for (int i=0; i<str.length-1; i++) {
                vals[i]= Integer.parseInt(str[i+1]);
                System.out.print(vals[i]+" ");
        }
      }
      else if(str[0].equals("stop")) {
            action=3;
      System.out.print( "Send> ");
      System.out.print("stop");
      m.set_action(action);
      }
      else if(str[0].equals("on")) {
            action=4;
      System.out.print( "Send> ");
      System.out.print("on"+" ");
            vals[0]=(int)0xffff;
            System.out.print("0xffff");
      }
      else if(str[0].equals("off")) {
            action=5;
            System.out.print( "Send> ");
      System.out.print("off"+" ");
          vals[0]=(int)0xffff;
            System.out.print("0xffff");
      }
      else if(str[0].equals("probe")) {
            action=6;
      System.out.print( "Send> ");
      System.out.print("probe");
      m.set_action(action);
      }
      else { usage(); }
      m.set_action(action);
    m.set_mpayload(vals);
    m_moteif.send( MoteIF.TOS_BCAST_ADDR, m );
    System.exit(0);
  }
  catch (IOException e)
  {
    e.printStackTrace();
    System.out.println("ERROR: Can't send message");
    System.exit(1);
  }
  catch (Exception e)
  {
      e.printStackTrace();
  }
}

public boolean sendIt()
{
      send( m_strings );
      return true;
}

public synchronized void messageReceived( int destAddr, Message m )
{
```

```
        /* nothing to do */
  }

  public static void main(String[] args)
  {
    Inject m = new Inject( args );
    m.sendIt();
  }
}
```

B.6 Makefile

Στην περίπτωση που η προσομοίωση γίνεται για το περιβάλλον TOSSIM, η μεταγλώττιση του κώδικα γίνεται με την εντολή make pc και η παράμετρος PFLAGS είναι απαραιτητη. Από την άλλη, σε περίπτωση μεταγλώττισης για αισθήτες mica2 (π.χ. για τον προγραμματισμό του αισθητήρα ή για λήψη μετρήσεων από το Avrora), εκτελούμε την εντολή make mica2 και η παράμετρος PFLAGS αφαιρείται.

```
PLATFORMS=mica mica2 pc
COMPONENT=Wrapper

PFLAGS += -I%T/platform/pc/CC1000Radio
include ../Makerules
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Discretix Technologies Ltd, *"Introduction to Side Channel Attacks"*. Available at: http://www.hbarel.com/publications/Introduction_To_Side_Channel_Attacks.pdf

[2] S.Glisic and B.Vucetic, *"Spread Spectrum CDMA Systems for Wireless Communications"*, Artech House. Boston, MA, (1997).

[3] A. Wood and J. Stancovic, *"Denial of Service in Wireless Sensor Networks"*, IEEE Computer, 35(10):54-62, (2002).

[4] G.Schafer, *"Sensor Network Security"*, The industrial Communications Technology Handbook, CRC Press, (2004).

[5] C. Karlof and D. Wagner, *"Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures"*, Elsevier's AdHoc Networks Journal, Special Issue on Sensor Network Applications and Protocols, (2003).

[6] A. Wood and J. Stankovic, "*Denial of Service in Sensor Networks*", IEEE Comput., Oct. (2002).

[7] D. Carman, P. Kruss and B. Matt, *"Constraints and Approaches for Distributed Sensor Network Security"*, NAI Labs Technical Report 00-010, (2000).

[8] N. Potlapally et al., "*Analyzing the Energy Consumption of Security Protocols*", ISLPED'03, Seoul, Korea, (2003).

[9] S. Basagni, K. Herrin, E. Rosti, and D. Bruschi. "*Secure Pebblenets*", In Proceedings of ACM MOBICHOC, (2001).

[10] S. Zhu, S. Setia and S. Jajodia, "*LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks*", In 10th ACM Conference on Computer and Communications Security, (2003).

[11] A. Perrig et al., "*SPINS: Security Protocols for Sensor Networks*", Mobile Network and Computing, (2001).

[12] C. Karlof, N. Sastry and D. Wagner, "*TinySec: A Link Layer Security Architecture for Wireless Sensor Networks*", SenSys'04, November 3-5, (2004).

[13] L. Eschenauer and V. Gligor, "*A Key-management Scheme for Distributed Sensor Networks*". Conf. on Computer and Communications Security. Proceedings of the 9th ACM conference on Computer and Communications Security, Washington, DC, USA, (2002).

[14] H. Chan, A. Perrig and D. Song, *"Random Key Predistribution Schemes for Sensor Networks"*. In: Proceedings of the IEEE Security and Privacy Symposium. IEEE Computer Society Press, pp. 197-213, Los Alamos (2003).

[15] D. Liu, and P. Ning, *"Establishing Pairwise Keys in Distributed Sensor Networks"*, Proceedings of the Conference on Computer and Communications Security '03, Washington DC, pp. 52-61, (2003).

[16] S. Schmidt et al, *"Security Architecture for Mobile Wireless Sensor Networks"*, ESAS 2004, C. Castelluccia et al. (eds.), LNCS 3313, pp. 166–177, (2005).

[17] G. Vernam, *"Cipher printing telegraph systems for secret wire and radio telegraphic communications"*, Journal of American Institution of Electronic Engineering, 45:252-259, (1926).

[18] C. Shannon, *"Communication Theory of Secrecy System"*, Bell Systems Technical Journal, 28:656-715, (1949).

[19] L. Foley and S. Wilson, *"Analysis of an On-line Random Number Generator"*, Trinity College Dublin, http://www.random.org, (2001).

[20] J. Walker, "*HotBits: Genuine Random Numbers Generated by Radioactive Decay*", (2006). Available at: http://www.fourmilab.ch/hotbits/.

[21] J. Lacy, D. Mitchell and W. Schell, "*CryptoLib: Cryptography in Software*", Proc 4th USENIX Security Symp, pg 1-17, (1993).

[22] J. Kelsey, B. Schneier and N. Ferguson, "*Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator*", Sixth Annual Workshop on Selected Areas in Cryptography, Springer Verlag, August (1999).

[23] Π. Κίκιρας "*Δίκτυα Αισθητήρων για Διάχυτο Υπολογισμό*", Διδακτορική Διατριβή ΕΜΠ, (2005).

[24] P. Levis, "*TinyOS Programming*", Revision 1.2, June (2006).

[25] D. Gay, P. Levis, D. Culler and E. Brewer, "*nesC 1.1 Language Reference Manual*", May (2003).

[26] B. Titzer, D. Lee and J. Palsberg, "*Avrora: Scalable Sensor Network Simulation with Precise Timing*". In Proceedings of IPSN'05, Fourth International Conference on Information Processing in Sensor Networks, Los Angeles, (2005).

[27] P. Levis, N. Lee, M. Welsh, and D. Culler, "*TOSSIM: Accurate and scalable simulation of entire TinyOS applications*", in Proceedings of SenSys'03, (2003).

[28] J. Polley, D. Blazakis, J. McGee, D. Rusk and J.S. Baras, "*ATEMU: a Fine-grained Sensor Network Simulator*", Sensor and Ad Hoc Communications and Networks, (2004).

[29] Crossbow Mica2 Datasheet, 6020-0042-07 Rev A. Available at: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf

[30] C. Karlof, N. Sastry and D. Wagner, "*TinySec: User Manual*", Berkley University, (2004).