# Σχεδιασμός και ανάπτυξη ασφαλούς IoT συστήματος εξουσιοδότησης με τεχνολογίες Mbed και RFID

**Θεοδώρα Τζιάστα**

**Διπλωματική Εργασία**

Επιβλέπων: Βασίλειος Τενέντες

Ιωάννινα, Μάρτιος 2023

**ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ**
**ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**
**UNIVERSITY OF IOANNINA**

# Ευχαριστίες

Ευχαριστώ πολύ τον κ. Βασίλειο Τενέντε τόσο για την υπομονή που έδειξε όλο αυτό το διάστημα όσο και για τις γνώσεις που μου προσέφερε. Ήταν πάντα διαθέσιμος να επιλύει οποιαδήποτε απορία προέκυπτε και είμαι ιδιαίτερα ευγνώμων γι' αυτό.

Ημερομηνία: 10 Μαρτίου 2023

Συγγραφέας: Θεοδώρα Τζιάστα

# Περίληψη

Η τεχνολογική εξέλιξη έχει οδηγήσει στην ανάπτυξη του Internet of Things ή Δίκτυο των Πραγμάτων. Η τεχνολογία αυτή έχει βρει εφαρμογή σε όλους τους τομείς της ανθρώπινης ζωής, αναθεωρώντας και την ασφάλεια χώρων με ειδικά σχεδιασμένα συστήματα εξουσιοδότησης και συγκεκριμένα συστήματα εξουσιοδοτημένης πρόσβασης σε χώρους. Αυτά τα συστήματα εγγυόνται ασφάλεια ελέγχοντας την είσοδο ενός ανθρώπου στο χώρο επιδεικνύοντας κάρτες πρόσβασης.

Πιο συγκεκριμένα, σκοπός της εργασίας είναι η υλοποίηση ενός ασφαλούς συστήματος εξουσιοδοτημένης πρόσβασης σε χώρους με κάρτες RFID, η κωδικοποιημένη επικοινωνία των IoT συσκευών με χρήση κρυπτογραφικών συναρτήσεων και ασφαλών πρωτοκόλλων ανταλλαγής κρυπτογραφικών κλειδιών καθώς και η αξιολόγηση της απόδοσης των κρυπτογραφικών διαδικασιών του συστήματος με ή χωρίς επιτάχυνση υλικού. Το σύστημα αποτελείται από έναν διακομιστή, τον μικροελεγκτή FRDM-K64F καθώς και το RFID σύστημα MIFARE RC522. Το λογισμικό του μικροελεγκτή βασίζεται στο λειτουργικό σύστημα πραγματικού χρόνου Mbed ενώ ο διακομιστής προγραμματίζεται σε Python. Το λογισμικό οδήγησης που χρησιμοποιείται για την επικοινωνία του μικροελεγκτή με το σύστημα RFID είναι το MFRC522.

Βασική ανάγκη του συστήματος εξουσιοδότησης είναι η ασφαλή ανταλλαγή πληροφοριών μεταξύ των IoT συσκευών προς αποφυγή κακόβουλων επιθέσεων. Έτσι, αναπτύσσεται υβριδικός κρυπτογραφικός αλγόριθμος συνδυάζοντας τον αλγόριθμο Rivest-Shamir-Adleman (RSA) για την ανταλλαγή των κρυπτογραφικών κλειδιών και τον αλγόριθμο Advanced Encryption Standard (AES) για την κρυπτογραφία των μηνυμάτων. Σε δεύτερη φάση για την ανταλλαγή κλειδιών υλοποιείται ο αλγόριθμος Diffie Hellman (DH). Οι παραπάνω υλοποιήσεις γίνονται κάνοντας χρήση της βιβλιοθήκης mbedtls και κρυπτογραφικών βιβλιοθηκών της Python.

Οι πιο συχνές, και άρα πιο χρονοβόρες διαδικασίες του συστήματος, είναι οι κλήσεις του AES. Έτσι, αξιολογούνται οι κρυπτογραφικές διεργασίες του AES συναρτήσει των διαφορετικών μεγέθους κλειδιών χρησιμοποιώντας αποκλειστικά λογισμικό αλλά και με επιτάχυνση υλικού ενεργοποιώντας τον συνεπεξεργαστή της πλατφόρμας με χρήση της βιβλιοθήκης mmCAU. Εξετάζεται η κατανάλωση ισχύος και ενέργειας, η ρυθμαπόδοση όπως και ο χρόνος αρχικοποίησης των διαφορετικών μεγέθους κλειδιών AES, ο χρόνος εκτέλεσης της κρυπτογράφησης και αποκρυπτογράφησης αλλά και οι

κύκλοι ρολογιού του επεξεργαστή ανά byte δεδομένων σε συνάρτηση του μεγέθους των κλειδιών του AES. Η βελτίωση που σημειώνεται κατά την αρχικοποίηση του κλειδιού AES είναι περίπου 70% με τη χρήση επιτάχυνσης υλικού. Ο χρόνος ολοκλήρωσης των υλοποιήσεων του AES βελτιώνεται από 0,0319ms με αποκλειστική χρήση λογισμικού σε 0,0212ms με χρήση του συνεπεξεργαστή. Η ρυθμαπόδοση σημειώνει 30% βελτίωση αφού κατά τη χρήση λογισμικού υπολογίζεται 0.47MB/s ενώ η αντίστοιχη τιμή με επιτάχυνση υλικού είναι 0.7 MB/s. Οι τιμές της κατανάλωσης ισχύος μπορεί να είναι από 0,701 έως 0,704 watt ενώ της ενέργειας που κυμαίνονται 0,020 έως 0,025 mJoule με τη χρήση αποκλειστικά λογισμικού και 0,015 έως 0,018 mJoule με τη χρήση επιτάχυνσης υλικού.

**Λέξεις Κλειδιά:** IoT, Σύστημα εξουσιοδότησης, Κρυπτογραφία

# Πίνακας περιεχομένων

# Κεφάλαιο 1. Εισαγωγή

Η πρόοδος της τεχνολογίας τα τελευταία χρόνια είναι ο βασικός λόγος που παρατηρείται γρήγορη αλλαγή στο μέγεθος, το σχήμα, την χωρητικότητα και την απόδοση των ηλεκτρονικών εξαρτημάτων. Οι κεντρικοί υπολογιστές έδωσαν τη σειρά τους στους ηλεκτρονικούς υπολογιστές που τους ακολούθησαν οι φορητοί υπολογιστές και οι φορητές συσκευές. Παρά το γεγονός της μείωσης του μεγέθους των συσκευών, πολλές φορές η βελτίωση της απόδοσης τους είναι σημαντική. Ταυτόχρονα με την βελτίωση του υλικού, η πρόοδος στα θέματα συνδεσιμότητας είναι εξίσου σημαντική. Οι απαιτήσεις των χρηστών που συνεχώς αυξάνονται σε συνδυασμό με τη σμίκρυνση των ηλεκτρονικών εξαρτημάτων κατέστησαν τις ηλεκτρονικές συσκευές αναπόσπαστο κομμάτι της καθημερινής ζωής. Στα πλαίσια της ανάγκης απλοποίησης, διευκόλυνσης και αυτοματοποίησης των καθημερινών αναγκών έρχεται να κάνει την εμφάνιση της η ιδέα του Internet of Things ή IoT όπως συναντάται συνήθως.

Η σύλληψη της ιδέας του IoT έγινε στα τέλη της δεκαετίας του 1990 αλλά η χρήση της τεχνολογίας ξεκίνησε περίπου το 2010 και εξελίσσεται ραγδαία αποτελώντας την τρίτη μεγάλη τεχνολογική εξέλιξη μετά τον υπολογιστή και το Διαδίκτυο. Το Internet of Things αναφέρεται σε ένα δίκτυο φυσικών ή ψηφιακών συσκευών όπου κάθε συσκευή έχει τη δυνατότητα ανταλλαγής δεδομένων με κάθε άλλη συσκευή του δικτύου. Τα δεδομένα αυτά αποστέλλονται και επεξεργάζονται από servers μέσω δικτύου, ετεροχρονισμένα ή σε πραγματικό χρόνο, με απώτερο σκοπό την άμεση και βέλτιστη λήψη αποφάσεων, εξ' αποστάσεως, ακόμα και χωρίς την παρέμβαση του ανθρώπου.

Η τεχνολογία αυτή έχει βρει εφαρμογή σε όλους τους τομείς την ανθρώπινης ζωής. Χρησιμοποιείται από την αυτοκινητοβιομηχανία με την ενσωμάτωση αισθητήρων στα αυτοκίνητα για την πρόβλεψη και αποφυγή προβλημάτων μέχρι και την αγροτική βιομηχανία με την χρήση αισθητήρων συλλογής δεδομένων της ατμόσφαιρας σε χωράφια ή μονάδες ζώων. Ακόμα μεγάλη εξέλιξη έχει επιφέρει και στον τομέα της

υγείας αφού με τη βοήθεια αισθητήρων που μπορούν να φορεθούν από τον ασθενή παρακολουθείται η υγεία του και βελτιώνεται η θεραπευτική του διαδικασία.

Το Internet of Things έχει γυρίσει σελίδα και στη ζωή μέσα στο σπίτι, μετατρέποντάς το σε «έξυπνο σπίτι». Κάθε παραδοσιακή συσκευή έχει μετατραπεί σε «έξυπνη» χρησιμοποιώντας αισθητήρες και συλλέγοντας συνεχώς δεδομένα εξατομικεύοντας την λειτουργία τους στις ανάγκες του εκάστοτε χρήστη. Μιλώντας για το «έξυπνο σπίτι» περιγράφεται ένα πλήρως εκσυγχρονισμένο οικιακό περιβάλλον όπου όλες οι συσκευές είναι συνδεδεμένες σε δίκτυο αυτοματοποιώντας και διευκολύνοντας τη καθημερινότητα αφού μπορούν να ελεγχθούν απομακρυσμένα. Παραδείγματα τέτοιων συσκευών μπορεί να είναι έξυπνη θερμοστάτες οι οποίοι με εξειδικευμένους αισθητήρες θερμοκρασίας κρατούν τη θερμοκρασία του σπιτιού πάντα στα επίπεδα που επιθυμεί ο χρήστης με απομακρυσμένη διαχείριση, έξυπνος θερμοσίφωνας ο οποίος μπορεί να λειτουργήσει ελεγχόμενος από εφαρμογή στο κινητό. Η καινοτόμα αυτή τεχνολογία γενικεύεται και σε χώρους μεγαλύτερων διαστάσεων όπως επιχειρήσεις και εργοστάσια. Εκτός από την διευκόλυνση των καθημερινών αναγκών εστιάζει στην ασφάλεια τους με ειδικά σχεδιασμένα συστήματα εξουσιοδότησης. Τα συστήματα αυτά μπορούν να χρησιμοποιηθούν τόσο για την εξουσιοδοτημένη πρόσβαση χρηστών σε εφαρμογές ή μηχανήματα όσο και για την ελεγχόμενη πρόσβαση ανθρώπων σε χώρους.

Οι παραδοσιακές κλειδαριές έχουν αντικατασταθεί με συστήματα εξουσιοδοτημένης πρόσβασης σε χώρους των οποίων η εξέλιξη είναι ραγδαία αφού η ασφάλεια αποτελεί βασική ανησυχία του σύγχρονου καταναλωτή. Συστήματα εξουσιοδότησης χαρακτηρίζονται τα συστήματα ασφαλείας που ελέγχουν την είσοδο ενός ανθρώπου σε ένα χώρο χρησιμοποιώντας κάρτες πρόσβασης. Κάποια από τα οφέλη που προσφέρουν τα συστήματα αυτά είναι ο έλεγχος του ατόμου που ζητά πρόσβαση σε ένα χώρο, την ώρα άφιξης και αποχώρησης του, το καθορισμό της πόρτας που δικαιούται να ξεκλειδώσει καθώς και υπό ποιες προϋποθέσεις θα του επιτραπεί η είσοδος. Τα περισσότερα συστήματα εξουσιοδότησης βασίζονται συστήματα RFID τα οποία καταφέρνουν ασύρματα και ανέπαφα με τη χρήση κυμάτων ραδιοσυχνοτήτων να ανιχνεύσουν αντικείμενα και να μεταφέρουν πληροφορίες. Ένα τέτοιο σύστημα αυτόματης εξουσιοδότησης χώρων είναι και το σύστημα που αναπτύσσεται στην παρούσα εργασία.

Σκοπός της εργασίας είναι η υλοποίηση ενός τέτοιου συστήματος εξουσιοδότησης με κάρτες RFID, η ασφαλή επικοινωνία των IoT συσκευών που το απαρτίζουν καθώς και η αξιολόγηση της απόδοσης του. Το σύστημα απαρτίζεται από έναν διακομιστή, τον μικροελεγκτή FRDM-K64F της εταιρείας NXP καθώς και το RFID σύστημα MIFARE

RC522 στο οποίο ανήκουν ο RFID αναγνώστης και οι RFID κάρτες. Ο μικροελεγκτής μαζί με τον αναγνώστη του RFID τοποθετούνται στην πόρτα ενώ οι RFID κάρτες δίνονται στους δυνητικούς χρήστες. Ο αναγνώστης του RFID δημιουργώντας ηλεκτρομαγνητικό πεδίο γύρω του, ανιχνεύει το σήμα που εκπέμπεται από την RFID κάρτα του χρήστη. Το μοναδικό αναγνωριστικό της κάρτας σε συνδυασμό με το αναγνωριστικό του μικροελεγκτή μεταφέρονται στον διακομιστή όπου ελέγχεται η άδεια εισόδου του χρήστη ή μη στο χώρο.

Πιο αναλυτικά, το λογισμικό του μικροελεγκτή FRDM-K64F βασίζεται στο λειτουργικό σύστημα πραγματικού χρόνου Mbed ενώ ο διακομιστής προγραμματίζεται σε Python. Το λογισμικό οδήγησης που χρησιμοποιείται για την επικοινωνία του μικροελεγκτή με το σύστημα RFID είναι το MFRC522.

Βασική ανάγκη του συστήματος εξουσιοδοτημένης πρόσβασης σε χώρους είναι η διασφάλιση της αναλλοίωτης ανταλλαγής πληροφοριών μεταξύ των IoT συσκευών προς αποφυγή κακόβουλων επιθέσεων. Έτσι, αναπτύσσεται υβριδικός κρυπτογραφικός αλγόριθμος συνδυάζοντας τον αλγόριθμο RSA για την ανταλλαγή των κρυπτογραφικών κλειδιών και τον αλγόριθμο AES για την κρυπτογραφία των μηνυμάτων. Σε δεύτερη φάση για την ανταλλαγή κλειδιών υλοποιείται ο αλγόριθμος Diffie Hellman έναντι του RSA. Οι παραπάνω υλοποιήσεις γίνονται κάνοντας χρήση της βιβλιοθήκης mbedtls και βιβλιοθηκών της Python.

Οι πιο συχνές, και άρα οι πιο χρονοβόρες διαδικασίες του συστήματος, είναι οι κλήσεις του AES, έτσι για την αξιολόγηση του συστήματος εξετάζονται οι κρυπτογραφικές διεργασίες του AES συγκρίνοντας δύο διαφορετικές μεθοδολογίες. Η πρώτη μεθοδολογία αφορά την απόδοση του συστήματος χρησιμοποιώντας αποκλειστικά λογισμικό ενώ στη δεύτερη εξετάζεται η απόδοσή του με επιτάχυνση υλικού. Για την επιτάχυνση υλικού ενεργοποιείται ο συνεπεξεργαστής της πλατφόρμας ο οποίος υποστηρίζει την υλοποίηση κρυπτογραφικών συναρτήσεων κάνοντας χρήση του υλικού της πλατφόρμας με σκοπό την βέλτιστη απόδοση του συστήματος. Εξετάζονται η κατανάλωση ισχύος και ενέργειας και η ρυθμαπόδοση (throughput) του συστήματος όπως και ο χρόνος αρχικοποίησης των διαφορετικών μεγέθους κλειδιών AES, ο χρόνος εκτέλεσης της κρυπτογράφησης και αποκρυπτογράφησης αλλά και οι κύκλοι ρολογιού του επεξεργαστή ανά byte δεδομένων συναρτήσει του μεγέθους των κλειδιών του AES. Η βελτίωση που σημειώνεται κατά την αρχικοποίηση του κλειδιού AES είναι περίπου 70% με τη χρήση επιτάχυνσης υλικού. Ο χρόνος ολοκλήρωσης των υλοποιήσεων του AES βελτιώνεται από 0,0319ms με αποκλειστική χρήση λογισμικού σε 0,0212ms με χρήση του συνεπεξεργαστή. Η ρυθμαπόδοση σημειώνει 30% βελτίωση αφού κατά τη

χρήση λογισμικού υπολογίζεται 0.47MB/s ενώ η αντίστοιχη τιμή με επιτάχυνση υλικού είναι 0.7 MB/s. Οι τιμές της ενέργειας που καταναλώνεται κυμαίνονται 0,020 έως 0,025 mJoule με τη χρήση αποκλειστικά λογισμικού και 0,015-0,018 mJoule με τη χρήση επιτάχυνσης υλικού.

Η δομή της εργασίας έχει ως εξής:

- Στο Κεφάλαιο 2 περιγράφονται οι τεχνολογίες που χρησιμοποιούνται καθώς και η βασική ορολογία που χρειάζεται να γνωρίζει ο αναγνώστης για το σύστημα πρόσβασης που έχει υλοποιηθεί.
- Στο Κεφάλαιο 3 αναλύεται εκτενώς η λειτουργία του συστήματος εξουσιοδότησης χώρων με RFID, ο αλγόριθμος που αναπτύχθηκε για την υλοποίηση του συστήματος καθώς και η διασύνδεση υλικού και λογισμικού που το αποτελούν .
- Στο Κεφάλαιο 4 αξιολογούνται οι κρυπτογραφικές διαδικασίες του αλγορίθμου AES με τη χρήση επιτάχυνσης υλικού ή αποκλειστικά με τη χρήση λογισμικού.

# Κεφάλαιο 2. Υπόβαθρο και Τεχνολογίες

Στο κεφάλαιο αυτό αναλύονται οι βασικές έννοιες, όπως το Δίκτυο των Πραγμάτων, η κρυπτογραφία και η επιτάχυνση υλικού. Ακόμα αναφέρονται πληροφορίες για την πλατφόρμα που επιλέχθηκε, για το λειτουργικό σύστημα Mbed όπως και για την τεχνολογία RFID. Ακόμα δίνονται λεπτομέρειες για τις βιβλιοθήκες mbedtls και mmCAU οι οποίες χρησιμοποιούνται για την ανάπτυξη του συστήματος.

## 2.1 Δίκτυο των πραγμάτων(IoT)

Με τον όρο Δίκτυο των πραγμάτων (Internet of things) ή IoT περιγράφεται η σύνδεση και η ανταλλαγή πληροφοριών μεταξύ πολλών και διαφορετικών ειδών συσκευών. Έχουν τη δυνατότητα σύνδεσης στο ίδιο τοπικό δίκτυο ή στο διαδίκτυο με σκοπό τον απομακρυσμένο έλεγχο τους. Τέτοιες συσκευές μπορεί να είναι οικιακές συσκευές, φώτα, κλιματιστικά, έξυπνα ρολόγια, αυτοκίνητα με ενσωματωμένους αισθητήρες αλλά και κάθε άλλη συσκευή που απαρτίζεται από λογισμικό, αισθητήρες και έχει την ικανότητα σύνδεσης σε δίκτυο[1].

Η λειτουργία ενός συστήματος IoT ξεκινά με τη συλλογή δεδομένων από το περιβάλλον τον αισθητήρων οι οποίο επικοινωνούν με έναν διακομιστή αποστέλλοντας τα δεδομένα. Στο διακομιστή γίνεται η επεξεργασία τους και έπειτα ειδοποιείται ο χρήστης και παρεμβαίνει εξ αποστάσεως όποτε χρειάζεται.

### 2.1.1 Ιστορία του «Δικτύου των πραγμάτων»

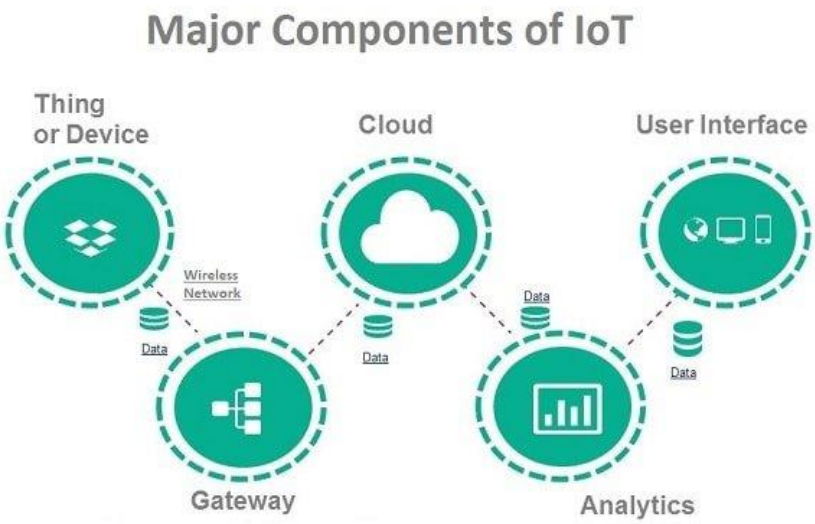
Η ιδέα του δικτύου των πραγμάτων πρωτοεμφανίστηκε στην αρχές του 1980 όπου μια ομάδα φοιτητών θέλησαν να ελέγξουν απομακρυσμένα το περιεχόμενο ενός αυτόματου

πωλητή Coca-Cola. Ο όρος Internet of Things εδραιώθηκε το 1999 από τον Kevin Aston ο οποίος πρότεινε την παρακολούθηση προϊόντων ενσωματώνοντας σε αυτά RFID chip.

Τις επόμενες δεκαετίες το ενδιαφέρον της τεχνολογίας στράφηκε στο IoT ξεκινώντας με την LG η οποία κυκλοφόρησε το πρώτο έξυπνο ψυγείο το 2000, με το πρώτο iPhone το 2007 ενώ η Google συνέχισε με τις δοκιμές των αυτοκινήτων χωρίς οδηγό.

Μέχρι το 2008 οι συνδεδεμένες συσκευές είχαν ξεπεράσει τον ανθρώπινο πληθυσμό. Μέχρι το 2025 41 δισεκατομμύρια συσκευές αναμένεται να συλλέγουν δεδομένα για τον τρόπο ζωής μας, τον τρόπο που δουλεύουμε και κινούμαστε στις πόλεις μας.

### 2.1.2 Μέρη ενός συστήματος IoT




*Εικόνα 1:Μέρη συστήματος IoT*

Ένα σύστημα IoT απαρτίζεται από έξι βασικά μέρη τα οποία είναι: συσκευές με ενσωματωμένους αισθητήρες (Devices or Things), οι πύλες (Gateways),η αποθήκευση τους στο Cloud, η επεξεργασία των δεδομένων (Data Processing-Analytics) και τέλος η διεπαφή χρήστη (User Interface). Όλα τα παραπάνω συνδέονται και ανταλλάσσουν δεδομένα μεταξύ τους.

Ξεκινώντας την περιγραφή των μερών του «Δικτύου των πραγμάτων» το ενδιαφέρον επικεντρώνεται στον όρο «Πράγματα» που αποτελεί και το κύριο μέρος του συστήματος. Στον όρο αυτό εμπεριέχονται οι αισθητήρες ή οποιαδήποτε φυσική συσκευή η οποία απαρτίζεται από αισθητήρες και χρησιμοποιείται για την συνεχή συλλογή δεδομένων από τον περιβάλλοντα χώρο των συσκευών. Τα δεδομένα μπορεί να είναι απλά, όπως η θερμοκρασία του χώρο αλλά μπορούν να γίνουν και αρκετά περίπλοκα, όπως ένα βίντεο. Κάποιοι από τους πιο συνηθισμένους αισθητήρες είναι

αισθητήρες θερμοκρασίας και υγρασίας, αισθητήρες μέτρησης καρδιακών παλμών, βημάτων και απόστασης, αισθητήρες ανίχνευσης κίνησης αλλά και RFID tags. Μία συσκευή IoT μπορεί να είναι ένα είδος αισθητήρα όπως κάμερα ή αισθητήρας θερμοκρασίας ή μια συσκευή να απαρτίζεται από διαφορετικά είδη αισθητήρων που λειτουργούν παράλληλα όπως ένα κινητό τηλέφωνο. Οι αισθητήρες μπορεί να βρίσκονται ακόμα και στις πιο δύσβατες περιοχές υπό κακές ή καλές καιρικές συνθήκες.

Οι παραπάνω συσκευές αφού συλλέξουν τα δεδομένα τα προωθούν στο επόμενο επίπεδο που ονομάζεται «Gateways». Το επίπεδο αυτό αποτελεί το κεντρικό σημείο όπου καταλήγουν τα δεδομένα πριν την αποστολή τους στο Cloud. Η αποστολή των δεδομένων μπορεί να γίνει μέσω Wi-Fi, Bluetooth, Ethernet με τη βοήθεια συσκευών όπως routers. Σκοπός αυτής της φάσης, εκτός από την λήψη των δεδομένων από τους αισθητήρες, είναι η υποβολή των δεδομένων σε μια πρώιμη επεξεργασία ώστε να μειωθεί ο θόρυβος κατά την αποστολή τους στο επόμενο επίπεδο, η προστασία τους εφαρμόζοντας πρωτόκολλα ασφαλείας, η επιλογή του πρωτοκόλλου επικοινωνίας και τέλος να τα στείλουν στην επόμενη φάση της επεξεργασίας τους, στα IoT Clouds.

Τα δεδομένα των αισθητήρων καταλήγουν σε IoT Clouds. Οι IoT Clouds είναι διασυνδεδεμένοι servers που έχουν σκοπό την συλλογή και επεξεργασία των δεδομένων σε υψηλές ταχύτητες, την διαχείριση της μετακίνησης τους και την αρχειοθέτηση τους.

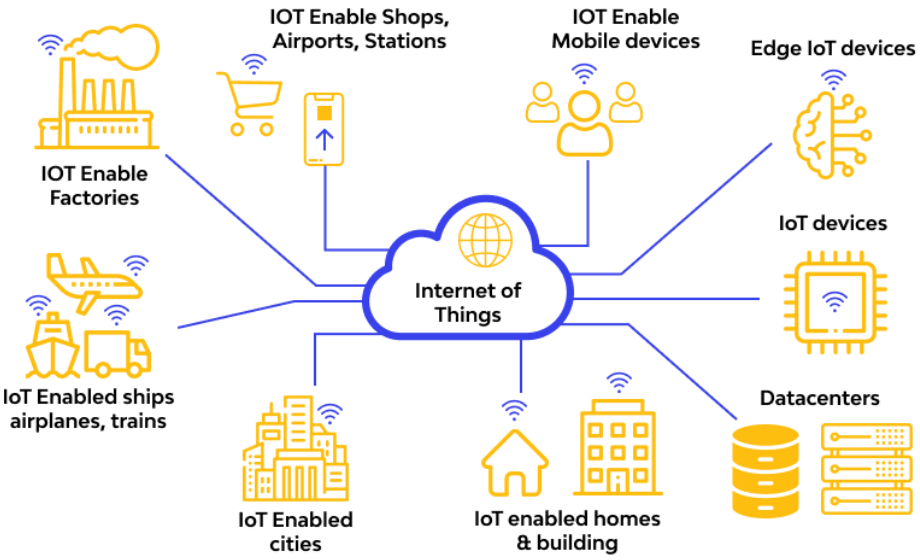
Η ανάλυση των δεδομένων γίνεται με την υλοποίηση αλγορίθμων και μπορεί να είναι απλή όπως ο κατά πόσο η τιμή της υγρασίας σε έναν χώρο είναι στα επιτρεπτά πλαίσια αλλά και περίπλοκη όπως για παράδειγμα εντοπισμός ενός εισβολέα στο σπίτι με χρήση αλγορίθμων υπολογιστικής όρασης.

Πολλές φορές μετά την ανάλυση των δεδομένων ίσως να χρειάζεται να ειδοποιηθεί ο χρήστη ή ακόμα να είναι απαραίτητη η δικιά του παρεμβολή ώστε να γνωστοποιήσει τις σκέψεις του σχετικά με τα αποτελέσματα ώστε να γίνουν οι κατάλληλες ενέργειες για την αποφυγή ανεπιθύμητων καταστάσεων. Για παράδειγμα εάν η τιμή της υγρασίας που βρέθηκε στο χώρο είναι σε πολύ υψηλά επίπεδα ίσως να προκαλέσει αρνητικές επιπτώσεις και η παρέμβαση του χρήστη είναι απαραίτητη για την διαμόρφωση της επόμενες ενέργειας.

Η παραπάνω ανάγκη οδηγεί στο τελευταίο επίπεδο του συστήματος IoT που αφορά τη διεπαφή με το χρήστη. Είναι συνήθως εφαρμογές προσαρμοσμένες στις ανάγκες του χρήστη δίνοντας του τη δυνατότητα να παρακολουθήσει τη ροή των δεδομένων μέσω μιας εφαρμογής στο κινητό ή στο web browser. Μπορεί να είναι απλές προειδοποιήσεις όπως ένα email ή μια ειδοποίηση στο κινητό ή άλλες φορές να δίνεται η δυνατότητα

στο χρήστη να παρέμβει τροποποιώντας την τιμή των δεδομένων και στέλνοντας πίσω την ενημέρωση κάνοντας την επικοινωνία αμφίδρομη[2].

### 2.1.3 Εφαρμογές του «Δικτύου των πραγμάτων»




*Εικόνα 2:Εφαρμογές IoT*

Παρόλο που η ιδέα του «δικτύου των πραγμάτων» υπάρχει εδώ και 30 χρόνια, η συχνότερη χρήση έχει αρχίσει την τελευταία δεκαετία. Η εν λόγω τεχνολογία βρίσκει εφαρμογή τόσο στο ευρύ φάσμα των επιχειρήσεων όσο στην καθημερινή ζωή του απλού πολίτη.

Το Internet of Things έχει αλλάξει ριζικά τη λειτουργία της βιομηχανίας καθώς συναντάται από τη γραμμή παραγωγής όσο μέχρι και την ασφάλεια ολόκληρης της εργοστασιακής μονάδας. Με τη χρήστη ενσωματωμένων αισθητήρων σε μηχανήματα, ράφια και χώρους μπορούν να ειδοποιηθούν άμεσα για προβλήματα καθιστώντας αποτελεσματικότερη και περισσότερο κερδοφόρα την επιχείρηση.

Στο χώρο της γεωργίας η ανάγκη για τον πλήρη έλεγχο των αγροτικών μονάδων και καλλιεργειών ανά πάσα στιγμή οδήγησε στην ενσωμάτωση αυτοματισμών στις αγροτικές δουλειές με δυνατότητα ελέγχου από απόσταση.

Σημαντική διευκόλυνση έχει επιφέρει και στον τομέα των μεταφορών. GPS αισθητήρες έχουν ενσωματωθεί στα αστικά λεωφορεία ελέγχοντας την θέση του λεωφορείου σε πραγματικό χρόνο και ενημερώνοντας για τον χρόνο άφιξης και αναχώρησης διευκολύνοντας έτσι την εξυπηρέτηση των πολιτών μειώνοντας την κυκλοφοριακή συμφόρηση στην πόλη. Ακόμη μπορεί να χρησιμοποιηθεί για έλεγχο των εργαζομένων σε ταξί και φορτηγά αλλά και σε περιπτώσεις κλοπής οχημάτων.

Ευεργετικές αλλαγές έχουν παρατηρηθεί και στον τομέα της υγείας με τη χρήση συσκευών IoT που τοποθετούνται στο περιβάλλον του ασθενή όπως αισθητήρες ανίχνευσης κίνησης αλλά και συσκευές που φοριούνται από τον ασθενή και ονομάζονται «wearables» όπως ένα έξυπνο ρολόι. Οι συσκευές αυτές περιέχουν ενσωματωμένους αισθητήρες μέτρησης καρδιακών παλμών, πίεσης, θερμοκρασίας, βημάτων. Η συλλογή αυτών των δεδομένων και η δυνατότητα απομακρυσμένης παρακολούθησής τους από τον γιατρό μπορεί να συμβάλει σημαντικά στην θεραπεία και αποκατάσταση της υγείας τους ασθενή.

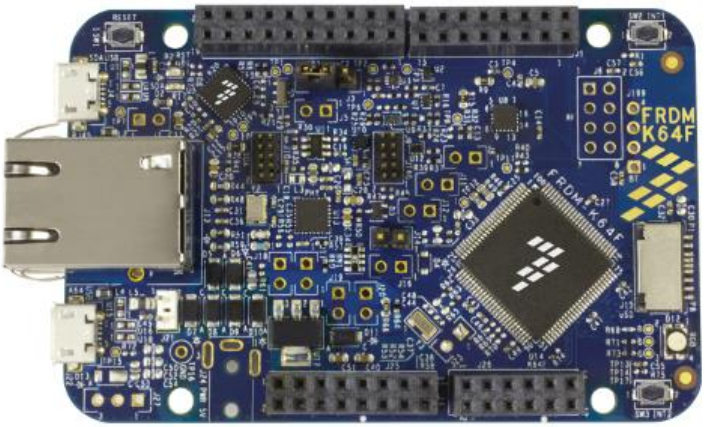
Το Internet of Things έχει φέρει πολλές καινοτομίες και στην καθημερινότητα του απλού πολίτη στο σπίτι δημιουργώντας τον όρο «Smart Home». Στον όρο «Smart Home» συμπεριλαμβάνονται οικιακές συσκευές στις οποίες έχουν ενσωματωθεί αισθητήρες των οποίων η λειτουργία μπορεί να ελεγχθεί εξ' αποστάσεως μέσω εφαρμογών. Οι συσκευές αυτές τείνουν να αντικαταστήσουν τις αντίστοιχες παραδοσιακές διευκολύνοντας την καθημερινότητα του ανθρώπου. Τέτοιες συσκευές είναι οι έξυπνες κλειδαριές, έξυπνες πρίζες, έξυπνες λάμπες και άλλες.

Κάποιες από τις διευκολύνσεις που προκύπτουν είναι ο απομακρυσμένος έλεγχος του θερμοσίφωνα ώστε να υπάρχει ζεστό νερό κατά την επιστροφή στο σπίτι, απομακρυσμένη ενεργοποίηση του πλυντηρίου και παρόμοιων συσκευών, απομακρυσμένη διαχείρισης λειτουργίας συσκευών στοχεύοντας στην εξοικονόμηση ενέργειας.

Εκτός από την μείωση της κατανάλωσης και την εξοικονόμηση ρεύματος, εξαλείφεται ο κίνδυνος πυρκαγιάς αφού μπορεί να ελεγχθεί απομακρυσμένα αν μια συσκευή παραμένει ανοιχτεί, προστατεύεται το σπίτι καθώς υπάρχει η δυνατότητα ενεργοποίησης του φωτισμού και των παντζουριών αυτόματα.

## 2.2 Πλατφόρμα

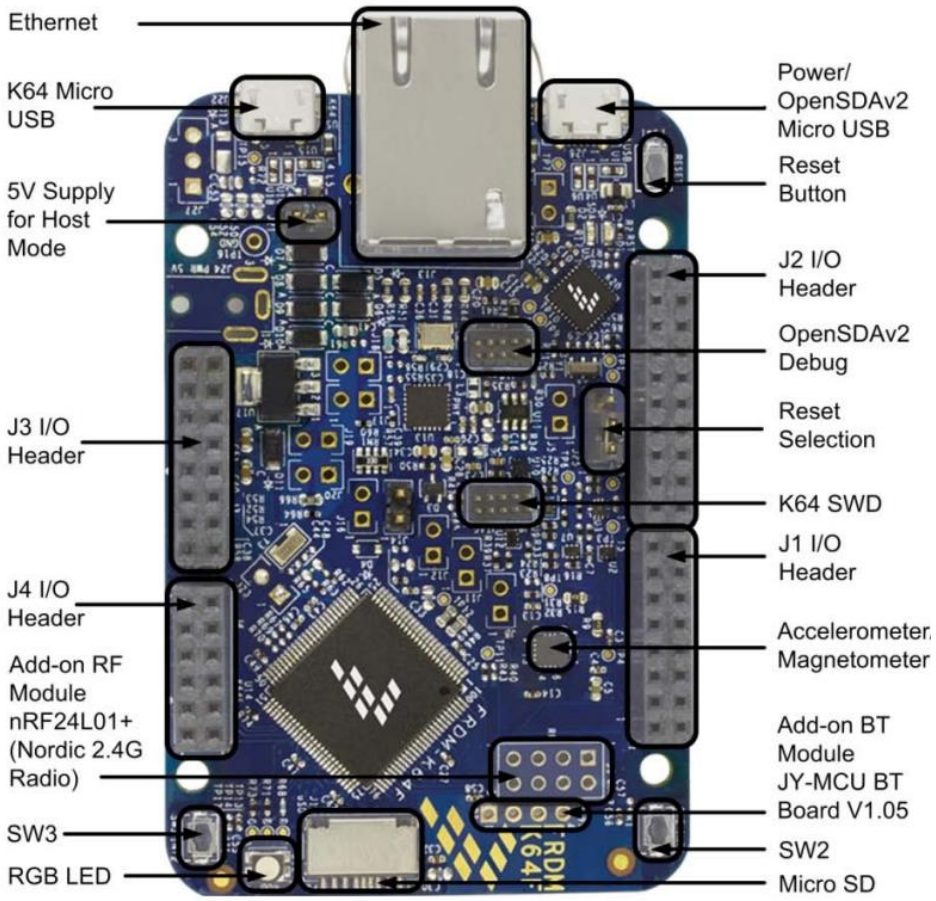
### 2.2.1 Εισαγωγή




*Εικόνα 3:Πλατφόρμα FRDM-K4F*

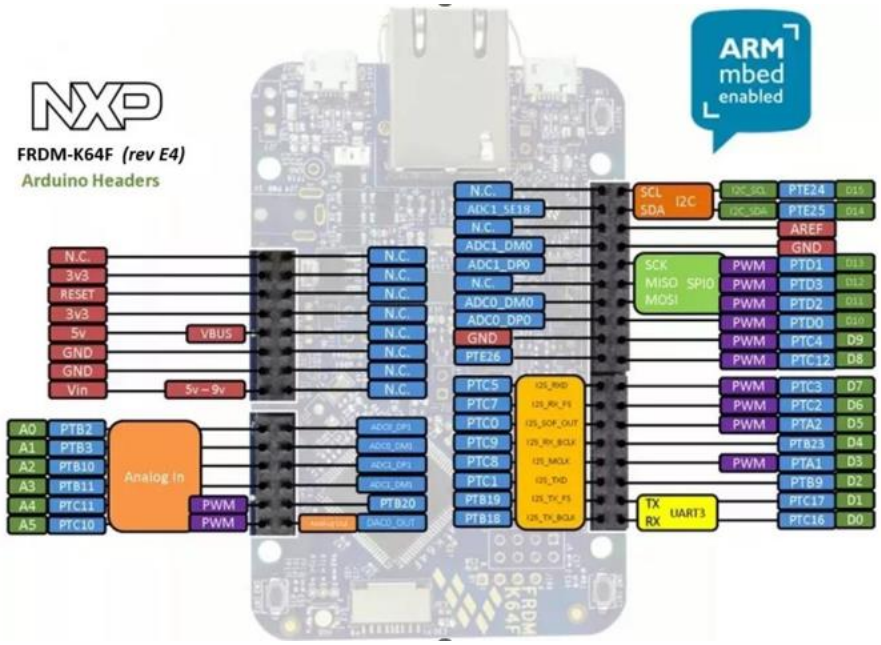
Το development board που χρησιμοποιήθηκε είναι το Freescale Freedom K64 ή εν συντομία FRDM-K64F. Έχει κατασκευαστεί από την εταιρεία NXP και ανήκει στην οικογένεια Kinetis K[3]. Είναι μια απλή και περιεκτική πλατφόρμα, ιδανική για εφαρμογές που βασίζονται σε microcontrollers που απαιτούν χαμηλή ισχύ αλλά και χαμηλό κόστος. Καθοριστικό χαρακτηριστικό ώστε να επιλεχθεί ως πλατφόρμα για το σύστημα που υλοποιείται είναι το γεγονός ότι υποστηρίζεται από το ανοιχτού τύπου λειτουργικό σύστημα της ARM Mbed OS για την κατασκευή εφαρμογών IoT. Η πλατφόρμα διαθέτει Ethernet interface και RF module. Τέλος, διαθέτει σον συνεπεξεργαστή Memory Mapped Crypto Acceleration Unit που αποτελεί βασικό κομμάτι του συστήματος.

### 2.2.2 Προδιαγραφές πλατφόρμας




*Εικόνα 4:Pins πλατφόρμας*

Στο board FRDM-K64F είναι τοποθετημένος ο microcontroller (MCU) MK64FN1M0VLL12. Έχει κατασκευαστεί από την εταιρεία NXP, ανήκει στην οικογένεια Kinetis K και πιο συγκεκριμένα στην σειρά K6x. Ο MCU διαθέτει 66 pins I/O τα οποία είναι συμβατά με αυτά του Arduino R3 και έτσι διευκολύνεται η διασύνδεση του microcontroller με άλλες περιφερειακές συσκευές. Ακόμα στο board βρίσκονται ενσωματωμένα δύο κουμπιά χρήστη και RGB LED.




*Εικόνα 5:Αντιστοιχία pins με Arduino*

### 2.2.2.1 Τροφοδοσία

Στο FRDM-K64F υπάρχουν διάφορες επιλογές τροφοδοσίας σε εύρος 1.71V - 3.3V. Στην παρούσα εργασία το board έχει τροφοδοτηθεί μέσω της θύρας micro-USB και διαρρέεται από τάση τροφοδοσίας 3.3V. Η συγκεκριμένη θύρα χρησιμοποιείται επίσης για την σύνδεση της πλατφόρμας με τον υπολογιστή καθώς και για τον προγραμματισμό της. Η σειριακή μεταφορά δεδομένων γίνεται με το πρωτόκολλο UART μέσω του pin μεταφοράς TX και λήψεις RX δεδομένων.

### 2.2.2.2 Επεξεργαστής

Η συσκευή βασίζεται στον επεξεργαστή ARM Cortex-M4. Είναι ένας υψηλής απόδοσης ενσωματωμένος επεξεργαστής 32-bit χαμηλού κόστους και χαμηλής κατανάλωσης. Λειτουργεί με μέγιστη συχνότητα 120MHz, διαθέτει 1 MB flash memory και 256 KB RAM. Προτιμάται σε αγορές που βασίζονται σε εφαρμογές IoT όπως της αυτοβιομηχανίας και του βιομηχανικού αυτοματισμού.

### 2.2.2.3 Ethernet

Η πλατφόρμα FRDM-K64F μπορεί να συνδεθεί σε τοπικό δίκτυο ή και διαδίκτυο αφού χαρακτηρίζεται από πρωτόκολλα Ethernet 10/100 MB/s. Το συγκεκριμένο

χαρακτηριστικό της πλατφόρμας ήταν καθοριστικής σημασίας καθώς η συνδεσμολογία Ethernet ήταν απαραίτητη για την υλοποίηση του συστήματος.

#### 2.2.2.4 RF Module

Ένα επίσης βασικό χαρακτηριστικό του FRDM-K64F είναι το γεγονός ότι διαθέτει headers για να υποστηρίζει RF radio modules. Για να μπορέσει να γίνει η μεταφορά των δεδομένων της περιφερειακής συσκευής προς τον microcontroller χρησιμοποιείται το πρωτόκολλο επικοινωνίας SPI (Serial Peripheral Interface).

#### 2.2.2.5 Crypto Acceleration Unit

Το FRDM-K64F διαθέτει εξειδικευμένη μονάδα κρυπτογραφικής επιτάχυνσης (Crypto Acceleration Unit ή CAU), η οποία έχει σχεδιαστεί αποκλειστικά για την επιτάχυνση της απόδοσης συναρτήσεων κρυπτογράφησης, αποκρυπτογράφησης και κατακερματισμού

Για να μπορέσει να εφαρμοστεί η επιτάχυνση υλικού στην πλατφόρμα FRDM-K64F χρησιμοποιήθηκε η βιβλιοθήκη ColdFire/ColdFire+ CAU and Kinetis mmCAU που έχει σχεδιαστεί από την εταιρεία NXP.

# 2.3 Τεχνολογία RFID

## 2.3.1 Εισαγωγή

Ο όρος RFID αποτελεί το ακρωνύμιο της φράσης Radio Frequency IDentification. Η απόδοση του όρου στα ελληνικά είναι «ταυτοποίηση μέσω ραδιοσυχνοτήτων»[4]. Πρόκειται για μια τεχνολογία η οποία χρησιμοποιεί ραδιοκύματα για να αναγνωρίζει και να παρακολουθεί ετικέτες (tags) που είναι ενσωματωμένες πάνω σε αντικείμενα. Η συγκεκριμένη τεχνολογία αποτελεί τον γενικό όρο των συστημάτων αυτόματου και ασύρματου εντοπισμού αντικειμένων ή ανθρώπων και είναι η εξέλιξη των ευρέως χρησιμοποιούμενων barcodes.

### 2.3.2 Μέρη ενός συστήματος RFID


*Εικόνα 6: Μέρη συστήματος RFID*

Ένα σύστημα RFID απαρτίζεται από δύο βασικά μέρη, τον πομποδέκτη (tag) και τον αναγνώστη (reader)[5].

#### 2.3.2.1 Πομποδέκτης

Ο πομποδέκτης, συνήθως αναφέρεται ως ετικέτα ή RFID κάρτα. Είναι μικρό chip, που το μέγεθός τους μπορεί να φτάσει και έως το 1/3 του χιλιοστού, μισός κόκκος άμμου. Οι RFID ετικέτες έχουν ενσωματωμένο έναν πομπό και έναν δέκτη. Ανάλογα με τον τρόπο επικοινωνίας μεταξύ των ετικετών και των αναγνωστών κατηγοριοποιούνται σε τρείς κατηγορίες που είναι οι εξής: Active Tags (ενεργές ετικέτες), Passive tags (παθητικές ετικέτες), και Semi-Passive Tags(ημι-παθητικές ετικέτες).

Το RFID tag, ανεξαρτήτως τον τύπο του, αποτελείται από δύο βασικά μέρη, το micro chip ή Integrated Circuit (IC) και μια κεραία (antenna). Έπειτα αναλόγως τον τύπο του tag προστίθενται κάποια επιπλέον στοιχεία.

Το micro chip ή Integrated Circuit (IC), είναι το ολοκληρωμένο κύκλωμα στο οποίο παίρνονται αποφάσεις και παρέχει μνήμη ώστε να αποθηκεύονται δεδομένα. Ανάλογα με τον τρόπο που έχει σχεδιαστεί μπορεί να περιέχει μνήμη read-only (RO), write-once-read many (WORM) και read-write (RW). Συνήθως τα tags χρησιμοποιούν electrically erasable, programmable, read-only memory (EEPROM). Επιπλέον σε ένα IC μπορεί να αποθηκευτούν μοναδικό αναγνωριστικό του tag, κωδικοί, ακόμα και κώδικες ανίχνευσης λάθους. Το μέγεθος της πληροφορίας που μπορεί να αποθηκευτεί σε αυτά είναι έως και 2KB. Τα ICs κατασκευάζονται σε wafer όπου το κάθε wafer μπορεί να περιέχει μέχρι και 40.000 ICs.

Το δεύτερο βασικό μέρος που απαρτίζει το RFID tag είναι η κεραία (antenna) που αποτελεί και το μεγαλύτερο μέρος του tag. Σκοπός της κεραίας είναι να απορροφά τα κύματα ραδιοσυχνότητας, να λαμβάνει και να μεταδίδει ή να αντανακλά (αναλόγως τον τύπο του tag) τα σήματα πίσω στον αναγνώστη. Η απόδοση των Passive tags εξαρτάται από το μέγεθος της κεραίας καθώς όσο μεγαλύτερη είναι, τόσο μεγαλύτερη ενέργεια μπορεί να συλλέξει το tag και στη συνέχεια να την στείλει πάλι πίσω, επομένως έχει μεγαλύτερο εύρος ανάγνωσης. Ακόμα το σχήμα μιας κεραίας μπορεί να ποικίλει και να δίνει εντελώς διαφορετικές δυνατότητες στο tag.

Τα επιπλέον μέρη που διαφοροποιούνται ανάλογα με τον τύπο του chip θα αναλυθούν παρακάτω.

#### 2.3.2.2 Παθητικές ετικέτες

Οι παθητικές ετικέτες αποτελούνται κυρίως από το Integrated Circuit (IC) και την κεραία, ενώ υπάρχουν ετικέτες που περιλαμβάνουν και το υπόστρωμα (substrate) .

Το τρίτο μέρος του passive tag είναι το υπόστρωμα, αρμοδιότητα του οποίου είναι να κρατά όλα τα μέρη του tag μαζί. Η κεραία τυπώνεται πάνω στο υπόστρωμα ενώ το IC προσδένεται πάνω σε αυτό. Το υπόστρωμα κατασκευάζεται από εύκαμπτα υλικά όπως λεπτό πλαστικό περίπου 100 έως 200nm.

Το passive tag περιμένει για ένα σήμα από τον αναγνώστη [6]. Ο αναγνώστης στέλνει ενέργεια στην κεραία η οποία μετατρέπει αυτή την ενέργεια σε κύμα ραδιοσυχνοτήτων που στέλνεται στη ζώνη ανάγνωσης του tag. Αυτή η ενέργεια μεταβιβάζεται από την κεραία του tag, στο IC όπου ενεργοποιείται το chip το οποίο με τη σειρά του δημιουργεί και στέλνει πίσω στο RF σύστημα ένα σήμα. Αυτό το σήμα γίνεται αντιληπτό από τον αναγνώστη, μέσω της κεραίας, και ερμηνεύει τα δεδομένα.

Η παθητική ετικέτα δεν διαθέτει μπαταρία, ούτε κάποιον άλλον εσωτερικό τρόπο τροφοδότησης αλλά τροφοδοτείται από την ενέργεια που μεταδίδεται από τον αναγνώστη. Αν και το εύρος του σήματος που αντιλαμβάνονται είναι πολύ μικρότερο από αυτό των active tags, το προσδόκιμο «όριο ζωής» τους είναι μεγαλύτερο. Χρησιμοποιούνται σε εφαρμογές όπως έλεγχος πρόσβασης, παρακολούθησης αρχείων κ.α.

Τα παθητικά tags λειτουργούν σε τρείς βασικές συχνότητες:

125 – 134 KHz – Low Frequency (LF): Μεγάλο μήκος κύματος με μικρό εύρος ανάγνωσης στα 1-10 εκατοστά. Αυτό το εύρος συχνοτήτων χρησιμοποιείται κυρίως στην παρακολούθηση ζώων.

13.56 MHz – High Frequency (HF) & Near-Field Communication (NFC): Μεσαίου μεγέθους μήκος κύματος αλλά μεγαλύτερο εύρος ανάγνωσης 1 εκατοστό έως 1 μέτρο. Χρησιμοποιείται για συστήματα μεταφοράς δεδομένων αλλά και ελέγχου πρόσβασης όπως αυτό που αναπτύσσεται στην παρούσα εργασία.

865 - 960 MHz – Ultra High Frequency (UHF): Μικρά αλλά υψηλής ενέργειας μήκη κύματος, κάποια passive tags έχουν εύρος ανάγνωσης 5-6 μέτρα, ενώ άλλα μπορούν να φτάσουν μέχρι και τα 30+ μέτρα. Αυτά τα tags χρησιμοποιούνται για παρακολούθηση αρχείων, διαχείριση πλυντηρίου.

Η RFID ετικέτα που έχει χρησιμοποιηθεί στο σύστημα εξουσιοδοτημένης πρόσβασης λειτουργεί σε συχνότητα 13.56MHz.

#### 2.3.2.3 Ενεργές ετικέτες

Οι ενεργές ετικέτες ή αλλιώς active tags παρά το γεγονός ότι απαρτίζονται από τα ίδια στοιχεία με αυτά των παθητικών ετικετών, τα micro chip συνήθως είναι μεγαλύτερα σε μέγεθος και έχουν περισσότερες δυνατότητες.

Σε αντίθεση με τις παθητικές ετικέτες, οι ενεργές ετικέτες έχουν ενσωματωμένες πηγές ενέργειας. Τέτοιες πηγές ενέργειας μπορεί να είναι κάποια μπαταρία ακόμα και κάποιος αισθητήρας ή microcontroller.

Ένα active tag είναι σε θέση συνεχώς να λαμβάνει σήματα και να μεταδίδει πληροφορίες που είναι αποθηκευμένες στο tag.

Τέτοιου τύπου ετικέτες λειτουργούν σαν Ultra High Frequency (UHF) passive tags καθώς οι συχνότητές τους είναι από 433MHz έως 960MHz και μπορούν να μεταδώσουν δεδομένα σε απόσταση έως και 150 μέτρα. Έτσι, οι ενεργές ετικέτες εφαρμόζονται σε

παρακολούθηση τοποθεσίας σε πραγματικό χρόνο, στα διόδια των αυτοκινητοδρόμων [6].

#### 2.3.2.4 Ημι-παθητικές ετικέτες

Οι ημι-παθητικές ετικέτες ή semi-passive tags είναι η ενδιάμεση επιλογή από τις ενεργές και τις παθητικές ετικέτες. Ενεργοποιείται μόνο υπό την παρουσία κάποιου αναγνώστη. Παίρνοντας χαρακτηριστικά και από τις δύο προαναφερθείσες κατηγορίες, χρησιμοποιούν ενσωματωμένη μπαταρία όπως φυσικά και κεραία και IC. Έχουν μικρότερο εύρος σήματος και ανάγνωσης συγκριτικά με αυτό των ενεργών ετικετών αλλά μεγαλύτερο εύρος ανάγνωσης από αυτό των παθητικών ετικετών. Είναι ιδανική επιλογή για συστήματα παρακολούθησης για το περιβάλλον.

2.3.2.5 Αναγνώστης

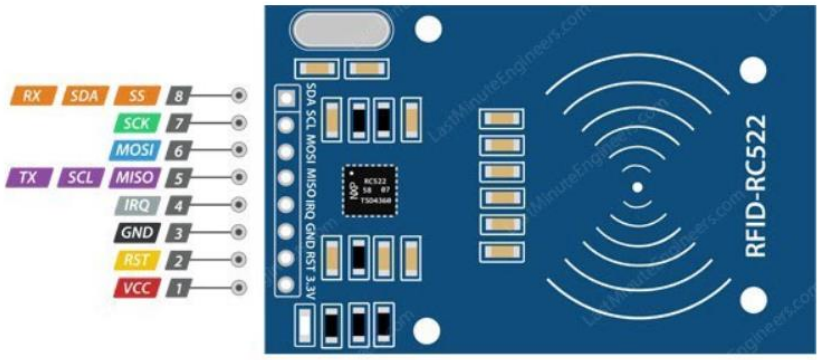
Ο αναγνώστης ή reader ή interrogator είναι μια συσκευή η οποία εκπέμπει σήμα μέσω μιας κεραίας. Αυτό το σήμα λαμβάνεται από το RFID tag το οποίο ανταποκρίνεται στον αναγνώστη όπως έχει προαναφερθεί. Ο αναγνώστης μετατρέπει την πληροφορία που του έχει σταλθεί σε ψηφιακό σήμα. Ανάλογα με τις εφαρμογές, τις διαστάσεις και τις τεχνικές ιδιότητες, οι αναγνώστες χωρίζονται σε σταθερούς, ολοκληρωμένους, αναγνώστες χειρός και ενσωματωμένους αναγνώστες.

### 2.3.3 RFID μονάδα

Η RFID μονάδα (module) που χρησιμοποιήθηκε για την διεκπεραίωση της παρούσας εργασίας είναι το Mifare RC522[7]. Το συγκεκριμένο module έχει κατασκευαστεί από την εταιρεία NXP και είναι ένα από τα οικονομικότερα που κυκλοφορούν στο εμπόριο. Συνήθως συνοδεύεται από δύο tags τα οποία έχουν 1KB μνήμης το κάθε ένα. Έχει την δυνατότητα να διαβάσει και να γράψει πληροφορία στην ετικέτα. Λειτουργεί δημιουργώντας ηλεκτρομαγνητικό πεδίο συχνότητας 13.56MHz ώστε να επικοινωνήσει ασύρματα με την ετικέτα. Τροφοδοτείται με τάση από 2.5V έως 3.3V. Για να μπορέσει να επικοινωνήσει με τον μικροελεγκτή χρησιμοποιείται Serial Peripheral Interface(SPI) πρωτόκολλο επικοινωνίας ενώ μπορεί να υποστηρίξει I2C και UART πρωτόκολλο.

### 2.3.4 Serial Peripheral Interface πρωτόκολλο

Το πρωτόκολλο επικοινωνίας Serial Peripheral Interface ή SPI είναι ένα σύγχρονο interface που ενεργοποιεί την σειριακή επικοινωνία μεταξύ του microcontroller και περιφερειακών συσκευών[8]. Χρησιμοποιεί διαφορετικές γραμμές μεταφοράς δεδομένων και σήμα ρολογιού καθιστώντας τις SPI συσκευές πλήρως συγχρονισμένες. Μια SPI συσκευή διαμορφώνεται ως αφέντης παράγοντας παλμούς ρολογιού ή ως σκλάβος και είναι υπεύθυνη για την αποστολή δεδομένων ύστερα από αίτημα του αφέντη. Σε ένα τέτοιο σύστημα αφέντη- σκλάβου δύναται να υπάρχουν περισσότεροι από έναν σκλάβοι. Στις περισσότερες περιπτώσεις ο microcontroller λειτουργεί ως αφέντης και οι περιφερειακές συσκευές ως σκλάβοι.




*Εικόνα 7:SPI πρωτόκολλο Pins*

Για να μπορέσουν να συνδεθούν οι δύο SPI συσκευές χρησιμοποιούνται τέσσερεις αρκοδέκτες οι οποίοι είναι οι εξής:

Master in, slave out-MISO: χρησιμοποιείται για την σειριακή μεταφορά δεδομένων από τον σκλάβο στον αφέντη. Το σήμα εξέρχεται από τον σκλάβο και εισέρχεται στον αφέντη.

Master out, slave in-MOSI: χρησιμοποιείται για την σειριακή μεταφορά δεδομένων από τον αφέντη στον σκλάβο. Το σήμα είναι έξοδος του αφέντη και καταλήγει ως είσοδος του σκλάβου.

Serial Clock-SCK: είναι το σήμα ρολογιού που χρησιμοποιείται για την συγχρονισμένη μεταφορά δεδομένων μεταξύ των SPI συσκευών. Παράγεται από τον αφέντη όπως έχει προαναφερθεί και αγνοείται από τους σκλάβους που δεν είναι ενεργοποιημένοι.

Chip Select-CS ή Slave Select-SS: το σήμα CS αποτελεί είσοδο της συσκευής σκλάβου. Στην περίπτωση που υπάρχουν περισσότερες περιφερειακές συσκευές, επιλέγεται ο επιθυμητός σκλάβος. Όταν το σήμα CS βρίσκεται σε κατάσταση low, τότε ο αφέντης επιλέγει τον σκλάβο που επιθυμεί για την μεταφορά δεδομένων.

### 2.3.5 Βιβλιοθήκη MFRC522

Για την λειτουργία του παραπάνω συστήματος έγινε χρήση της Arduino βιβλιοθήκης MFRC522 η οποία είναι συμβατή με το FRDM-K64F.

Η βιβλιοθήκη προσφέρει τη δυνατότητα ανάγνωσης RFID καρτών με τον Mifare RC522 αναγνώστη χρησιμοποιώντας το SPI πρωτόκολλο επικοινωνίας. Το μοναδικό αναγνωριστικό της κάρτας δεν μπορεί να τροποποιηθεί επομένως υπάρχουν ζητήματα που πρέπει να ληφθούν υπόψιν στο θέμα της ασφάλειας.

Κατά την υλοποίηση της εργασίας χρησιμοποιήθηκαν τα αρχεία MFRC522.h και MFRC522.cpp της βιβλιοθήκης.

Η βιβλιοθήκη παρέχει συναρτήσεις για την ανίχνευση νέων συσκευών που συναντόνται στο ηλεκτρομαγνητικό πεδίο γύρω από τον RFID αναγνώστη ενώ με την συνάρτηση PICC_ReadCardSerial() διαβάζεται ο αριθμός μίας από τις κάρτες που έχουν ανιχνευτεί.

Η συντομογραφία PCD που συναντάται συχνά στα ονόματα των συναρτήσεων της βιβλιοθήκης αποτελεί ακρωνύμιο της φράσης Proximity coupling device. Αναφέρεται σε συσκευές που έχουν την ικανότητα να διαβάσουν ετικέτες με βάση το ISO όπως για παράδειγμα ο RFID reader. Από την άλλη, η PICC είναι συντομογραφία της Proximity inductive coupling card και απευθύνεται σε συσκευές που μπορούν να διαβαστούν ή να γραφτούν από έναν αναγνώστη ενώ δε διαθέτουν πηγή τροφοδοσίας αλλά τροφοδοτούνται από το ηλεκτρομαγνητικό πεδίο του αναγνώστη όπως ακριβώς οι RFID κάρτες που χρησιμοποιούνται στο σύστημα εξουσιοδότησης.

## 2.4 ARM Mbed OS

### 2.4.1 Εισαγωγή

Για την υλοποίηση του IoT συστήματος επιλέχθηκε το λειτουργικό σύστημα ARM mbed[9]. Το mbed OS είναι ένα ανοιχτού τύπου λογισμικό πραγματικού χρόνου το οποίο αναπτύχθηκε στα τέλη του 2009 από την εταιρεία ARM και συνεχίζει να ενημερώνεται μέχρι και σήμερα έχοντας φτάσει στην 6η του έκδοση. Βασίζεται σε πλατφόρμες που χρησιμοποιούν τον 32-bit Arm Cortex-M επεξεργαστή. Χρησιμοποιείται σε IoT εφαρμογές, χαμηλής κατανάλωσης και ισχύος.

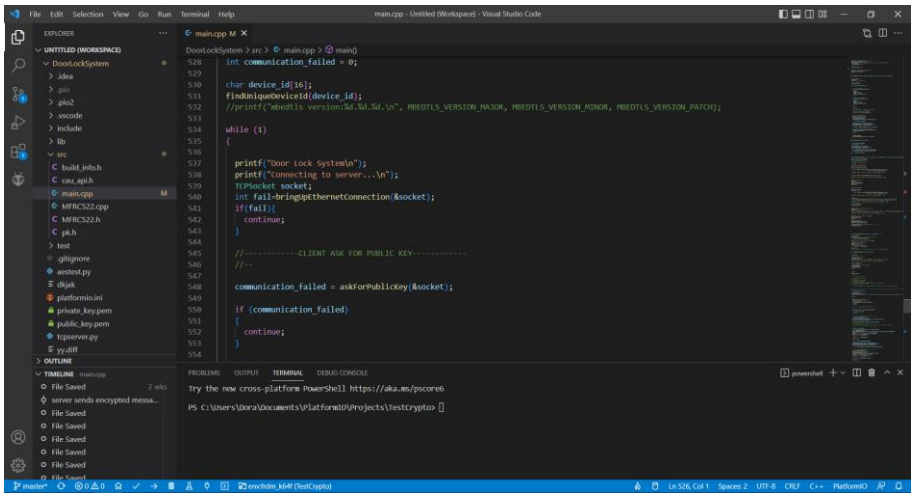
Είναι μια πλατφόρμα που παρέχει ένα πλήρες Application Programming Interface (API) ώστε να αναπτύξει εύκολα κανείς IoT εφαρμογές. Το mbed OS υποστηρίζει μια πληθώρα βιβλιοθηκών και οδηγών που δίνουν τη δυνατότητα στον microcontroller να επικοινωνήσει με περιφερειακές συσκευές όπως αισθητήρες και RFID. Επιπλέον παρέχει επιλογές δικτύωσης όπως Ethernet και Wi-Fi ενώ μπορεί να εγγυηθεί ασφαλή διασύνδεση υποστηρίζοντας Mbed TLS βιβλιοθήκες.

Το λειτουργικό σύστημα δίνει την δυνατότητα στους προγραμματιστές να αναπτύξουν εύκολα κάθε είδους εφαρμογή που βασίζεται σε microcontrollers γράφοντας σε C ή C++[10]. Αυτό το καταφέρνουν αφού στο λειτουργικό σύστημα mbed παρέχεται ένα αφαιρετικό επίπεδο (abstraction layer) ανάμεσα στο software και στο hardware. Εξίσου σημαντικό είναι πως το software που έχει γραφτεί μπορεί να μεταφερθεί σε οποιαδήποτε άλλη πλατφόρμα που υποστηρίζεται από το mbed OS.

Υπάρχουν αρκετά περιβάλλοντα ανάπτυξης mbed εφαρμογών. Κάποια από αυτά είναι πιο εξειδικευμένα Integrated Development Environments (IDEs) όπως το Mbed Online Compiler το οποίο όπως υποδηλώνει και το όνομα του είναι online, το Mbed Studio και το Mbed CLI τα οποία είναι offline με την διαφορά ότι το δεύτερο δεν διαθέτει γραφικό περιβάλλον αλλά περιβάλλον εντολών. Τα παραπάνω εργαλεία χρησιμοποιούν τον ARMCC C/C++ compiler και παρέχουν την ευχέρια στον προγραμματιστεί να ανάπτυξη την εφαρμογή που επιθυμεί έχοντας στη διάθεση του μια πληθώρα βιβλιοθηκών.

Επιπλέον οι mbed εφαρμογές εκτός από τα παραπάνω εξειδικευμένα IDEs που αναφέρθηκαν μπορούν να αναπτυχθούν και σε γενικά περιβάλλοντα ανάπτυξης όπως το Eclipse, το Keil μVision και το PlatformIO, ένα extension του Visual Studio Code IDE στο οποίο έγινε και η πλήρης ανάπτυξη της συγκεκριμένες εφαρμογής.

### 2.4.2 Προγραμματιστικό περιβάλλον PlatformIO



*Εικόνα 8:Προγραμαμτιστικό περιβάλλον PlatformIO*

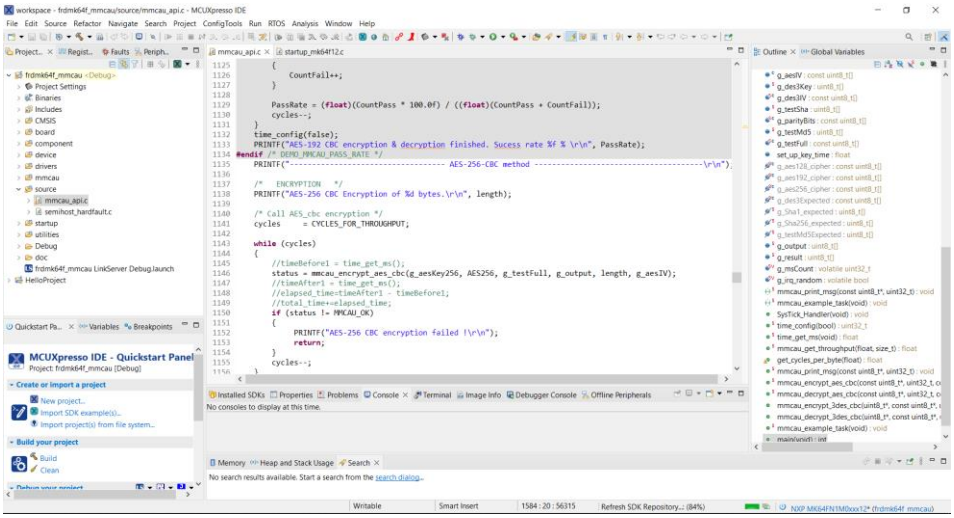
Η σχεδίαση και ανάπτυξη της εφαρμογής έγινε εξολοκλήρου στον προγραμματιστικό περιβάλλον PlatformIO που αποτελεί extension του IDE Visual Studio Code[11] . Είναι ένα ανοιχτού τύπου λογισμικού IDE που εξειδικεύεται στην ανάπτυξη embedded εφαρμογών, υποστηρίζει μεγάλη συλλογή από πλατφόρμες όπως Espressif, STM32 και φυσικά Freescale Kinetis ενώ παρέχει και πληθώρα frameworks όπως mbed, Arduino, Zephyr.

Το PlatformIO έχει προγραμματιστεί σε Python, μπορεί να υποστηρίξει plugins που διευκολύνουν και επιτυγχάνουν την διαδικασία προγραμματισμού της εφαρμογής με προσαρμοσμένο γραφικό περιβάλλον στις ανάγκες του προγραμματιστεί. Ενσωματώνει τερματικό PlatformIO Core (CLI) και σειριακή οθόνη (Serial Monitor).

Το IDE παρέχει μια πλήρης λύση debugging για εύκολο εντοπισμό σφαλμάτων απλοποιώντας τις περίπλοκες διαδικασίες κάνοντάς τες με αυτοματοποιημένο τρόπο. Το PlatformIO Unit Testing εστιάζοντας και δοκιμάζοντας τη λειτουργία μεμονωμένα για κάθε μέρος της εφαρμογής βοηθά στην μείωση των σφαλμάτων που πιθανότατα θα προκύψουν.

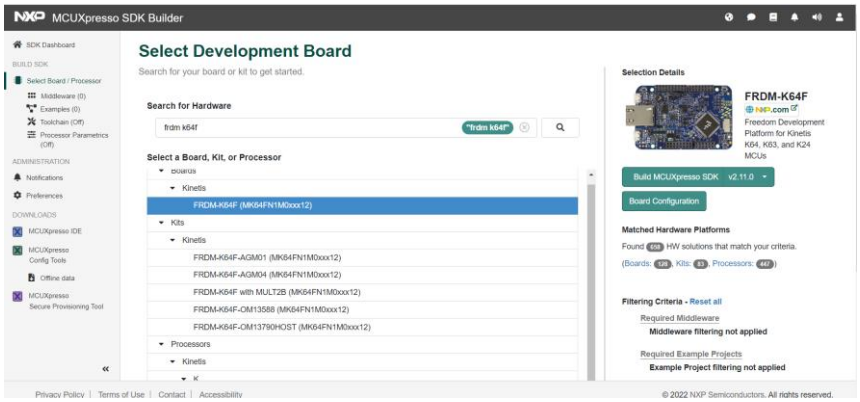
Ένα από τα καινοτόμα χαρακτηριστικά του είναι η δυνατότητα εξ αποστάσεως προγραμματισμού, testing ή debugging σε συσκευές οπουδήποτε στον κόσμο, ενώ αυτό που το κάνει να ξεχωρίζει είναι η ευελιξία που δίνει στον προγραμματιστή ώστε να μεταγλωττίσει τον ίδιο κώδικα σε διαφορετικά boards. Αυτό πραγματοποιείται ρυθμίζοντας κατάλληλα για κάθε πλατφόρμα το Project Configuration File, platformio.ini.

### 2.4.3 MCUXpresso IDE και MCUXpresso SDK Builder



*Εικόνα 9: Περιβάλλον MCUXpresso*

Το MCUXpresso είναι ένα περιβάλλον ανάπτυξης εφαρμογών για μικροεπεξεργαστών της εταιρείας NXP που βασίζονται στον επεξεργαστή ARM της σειράς Cortex-M. Το MCUXpresso IDE αποτελεί μια πλήρη λύση σε προγραμματιστές που θέλουν να αναπτύξουν embedded εφαρμογές προσφέροντάς τους την δυνατότητα επεξεργασίας, compile και debug. Το IDEβρίσκεται στην 11η έκδοση του και έχει δημιουργηθεί βασισμένο πάνω στο προγραμματιστικό περιβάλλον Eclipse.




*Εικόνα 10: Διαδικασία επιλογής SDK*

Στο MCUXpresso IDE βρίσκεται ενσωματωμένο το MCUXpresso Software Development Kit (SDK) Builder προσφέροντας ολοκληρωμένα πακέτα επιτυγχάνοντας την διαδικασία σχεδιασμού και ανάπτυξης της εφαρμογής. Η NXP παρέχει drivers, λογισμικό, παραδείγματα βιβλιοθηκών για κάθε development board που έχει σχεδιάσει

με τον ARM Cortex-M επεξεργαστή. Τα zip αρχεία με drag&drop εγκαθίστανται στο IDE[12].

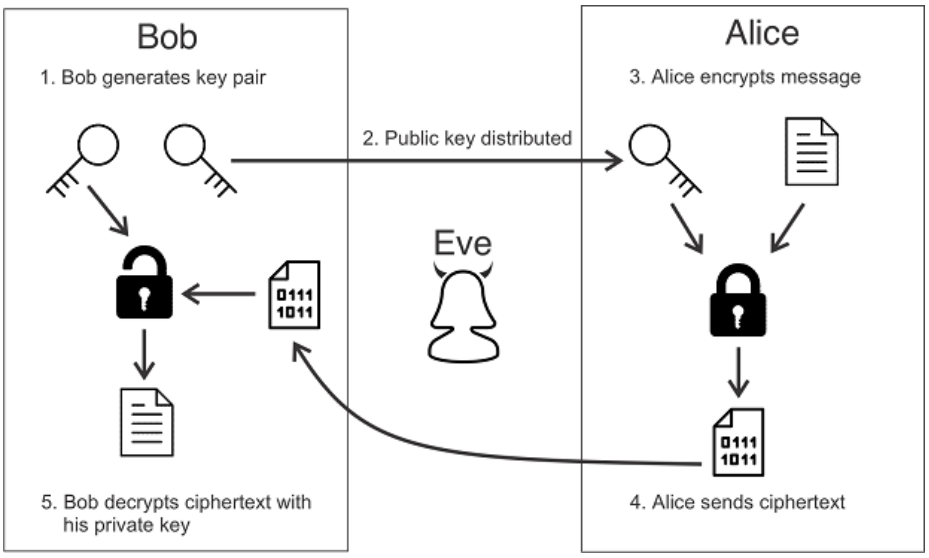
# 2.5 Κρυπτογραφία

## 2.5.1 Εισαγωγή

Ακόμα και σε αρχαιότερες εποχές, πριν την ψηφιοποίηση της πληροφορίας, υπήρχε η ανάγκη προστασίας και διασφάλισης της μυστικότητας των δεδομένων που ανταλλάσσονταν κατά την επικοινωνία των μερών ενός συστήματος. Έτσι αναπτύχθηκε η επιστήμη της κρυπτογραφίας κατά την οποία σκόπιμα αλλοιώνεται η αρχική πληροφορία και παράγεται ένα κρυπτογραφημένο κείμενο αποτρέποντας έτσι την διαρροή των δεδομένων σε τυχόν επιτήδειους. Ιδανικά μόνο ο αποστολέας και ο παραλήπτης διαθέτουν τις τεχνικές ώστε να αποκρυπτογραφηθεί το κρυπτοκείμενο και να παραχθεί ξανά η αρχική πληροφορία.

Βασική διαδικασία της κρυπτογραφίας είναι η κρυπτογράφηση κατά την οποία ο αποστολέας με συγκεκριμένο κλειδί κρυπτογραφεί την πληροφορία που πρόκειται να αποσταλεί στο δίκτυο ενώ κατά την διαδικασία της αποκρυπτογράφησης ο παραλήπτης με εξειδικευμένο μαθηματικό αλγόριθμο παράγει και πάλι το αρχικό κείμενο[13].

### 2.5.2 Αλγόριθμος RSA




*Εικόνα 11: Αλγόριθμος RSA*

Ο αλγόριθμος RSA προτάθηκε για πρώτη φορά το 1976 από τους Rivest, Shamir και Adleman από τους οποίους πήρε και το όνομα του. Είναι ένας αλγόριθμος κρυπτογράφησης που ανήκει στην κατηγορία αλγορίθμων ασύμμετρου κλειδιού. Ακόμα χρησιμοποιείται για ανταλλαγή κλειδιών και ψηφιακή υπογραφή. Κάθε κόμβος που συμμετέχει στην επικοινωνία διαθέτει ένα ζευγάρι ιδιωτικού και δημόσιου κλειδιού. Το δημόσιο κλειδί είναι γνωστό σε όλους και χρησιμοποιείται για την διαδικασία της κρυπτογράφησης ενώ το ιδιωτικό κλειδί είναι γνωστό μόνο στον κάτοχο του και χρησιμοποιείται για την αποκρυπτογράφηση του μηνύματος.

Όπως φαίνεται και από την παραπάνω εικόνα, όταν ο Α θελήσει να στείλει ένα κρυπτογραφημένο μήνυμα στον Β, τότε χρησιμοποιώντας το δημόσιο κλειδί του Β, το οποίο είναι γνωστό σε όλους κρυπτογραφεί το μήνυμα. Ωστόσο, μόνο ο Β μπορεί να αποκρυπτογραφήσει το μήνυμα αφού είναι ο μοναδικός που διαθέτει το κατάλληλο ιδιωτικό κλειδί.

Η ασφάλεια του αλγορίθμου εναπόκεινται στο γεγονός ότι τα δημόσια και ιδιωτικά κλειδιά είναι συναρτήσεις ενός ζεύγους μεγάλων πρώτων αριθμών έως 2048 bits[14].

### 2.5.3 Αλγόριθμος AES

Το 1997 προτάθηκε ο αλγόριθμός AES, το οποίο αποτελεί ακρωνύμιο της φράσης Advanced Encryption Standard. Είναι ένας αλγόριθμος συμμετρικής κρυπτογράφησης block. Το αρχικό κείμενο χωρίζεται σε τμήματα των 128-bit και χρησιμοποιείται το ίδιο

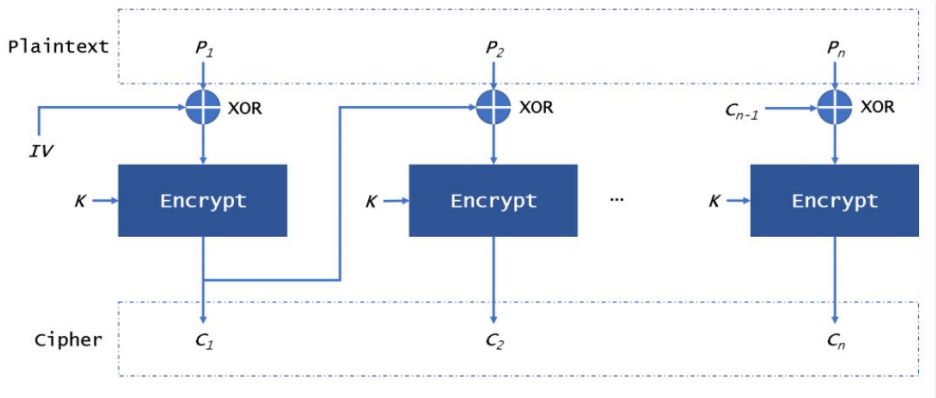
κλειδί κατά την κρυπτογράφηση και αποκρυπτογράφηση. Το μέγεθος του κλειδιού μπορεί να διαφέρει σε 128, 192 ή 256 bits.

Ο αλγόριθμος εφαρμόζει πολλαπλούς κύκλους (rounds) κατά την διάρκεια της κρυπτογράφησης των δεδομένων. Το κλειδί μεγέθους 128-bit απαιτεί 10 κύκλους, το κλειδί των 192-bit απαιτεί 12 κύκλους ενώ το κλειδί μεγέθους 256-bit απαιτεί 14 κύκλους. Κάθε ένας από τους γύρους χρειάζεται ένα στρογγυλό κλειδί που είναι επέκταση του αρχικού κλειδιού. Αυτοί οι γύροι καθιστούν τον AES τόσο ισχυρό.

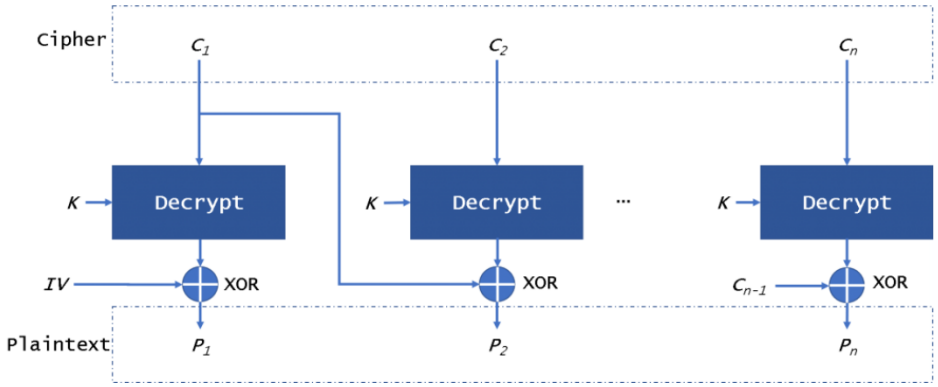
Ο AES είναι ένας από τους σημαντικότερους αλγορίθμους κρυπτογράφησης σήμερα καθώς είναι γρήγορος και ασφαλής [15].

Για την περίπτωση πολλών block έχουν αναπτυχθεί τέσσερεις διαφορετικοί τύποι της λειτουργίας του AES, ECB, CBC,CFB,OFB,CTR. Για τους σκοπούς του συστήματος χρησιμοποιήθηκε ο τύπος CBC[16].

### 2.5.3.1 CBC mode




*Εικόνα 12: Κρυπτογράφηση AES CBC*




*Εικόνα 13: Αποκρυπτογράφηση AES CBC*

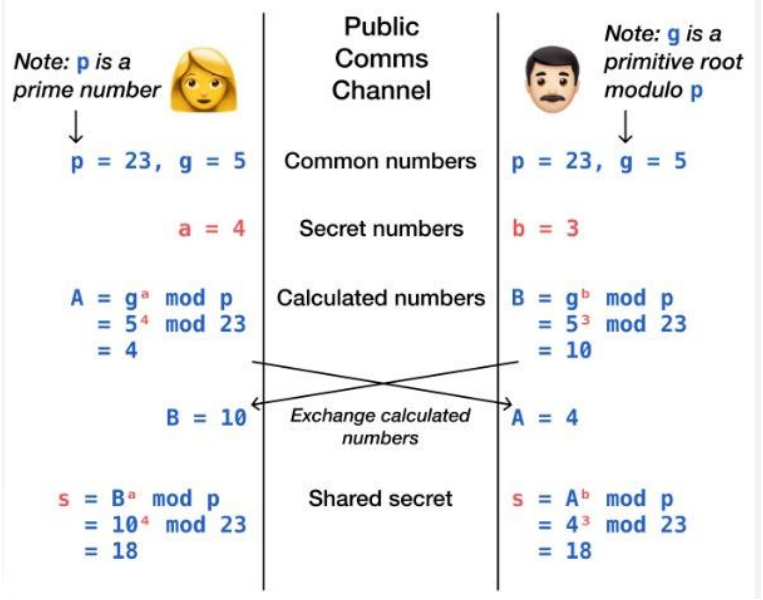
Το Cipher Block Chaining mode αποτελεί έναν τύπο κρυπτογράφησης AES και εκτός από το κλειδί της κρυπτογράφησης χρησιμοποιεί ένα ακόμα διάνυσμα αρχικοποίησης ίδιου μεγέθους με το τμήμα κρυπτογράφησης. Μεταξύ κάθε τμήματος του αρχικού κειμένου και του διανύσματος αρχικοποίησης γίνεται μια πράξη XOR και έπειτα με το κλειδί της κρυπτογράφησης προκύπτει το κρυπτοκείμενο. Στο επόμενο τμήμα του αρχικού κειμένου θα γίνει XOR με προηγούμενο κρυπτοκείμενο μέχρι το τελευταίο block.

Κάθε στοιχείο του κρυπτογραφημένου κειμένου εξαρτάται από το αντίστοιχο στοιχείο του αρχικού κειμένου αλλά και από όλα τα προηγούμενα τμήματα του αρχικού κειμένου. Αν αλλάξουμε την σειρά των κρυπτοκειμένων καταστρέφεται η διαδικασία της αποκρυπτογράφησης. Αυτή η συμπεριφορά καθιστά την κρυπτογράφηση των τμημάτων αλυσιδωτή από την οποία προκύπτει και το όνομα του τύπου.

Η αποκρυπτογράφηση μπορεί να γίνει παράλληλα αν όλα τα κρυπτογραφημένα block είναι διαθέσιμα[17].

### 2.5.4 Αλγόριθμος Diffie Hellman

Ο αλγόριθμος Diffie Hellman(DH) είναι ένα πρωτόκολλο ανταλλαγής κλειδιών που επιτρέπει στον αποστολέα και στον παραλήπτη να καταλήξουν στο ίδιο μυστικό κλειδί χωρίς αυτό να μεταφερθεί πάνω από το δίκτυο. Προτάθηκε αρχικά το 1976 και χρησιμοποιείται για την ανταλλαγή κλειδιών συμμετρικής κρυπτογράφησης.




*Εικόνα 14: Αλγόριθμος Diffie Hellman*

Όπως φαίνεται και από το παραπάνω σχήμα, τα δύο άκρα συμφωνούν στη χρήση δύο κοινών αριθμών P,G οι οποίοι μπορούν να μεταφερθούν πάνω από το δίκτυο χωρίς

κίνδυνο. Ο P είναι ένας πρώτος αριθμός και ο G είναι πρώτη ρίζα του P. Έπειτα ο κάθε ένας επιλέγει ένα τυχαίο ιδιωτικό αριθμό. Ακολουθούν μαθηματικοί υπολογισμοί που καταλήγουν σε αριθμούς A,B που ανταλλάζονται πάνω από το δίκτυο. Με βάση τους αριθμούς A,B και P τα δύο άκρα καταλήγουν στο ίδιο μυστικό κλειδί χωρίς αυτό να έχει μεταφερθεί πάνω από το απροστάτευτο κανάλι επικοινωνίας.

Το μυστικό κλειδί που διαθέτουν τα άκρα μπορεί να χρησιμοποιηθεί για κρυπτογράφηση και αποκρυπτογράφηση με κατάλληλους αλγορίθμους όπως ο AES[18].

### 2.5.5 Υβριδικό κρυπτοσύστημα

Κατά γενικό κανόνα το υβριδικό κρυπτοσύστημα χρησιμοποιεί έναν μηχανισμό δημόσιου κλειδιού όπως DH ή αλγόριθμο ασύμμετρου κλειδιού όπως ο RSA και έναν αλγόριθμο συμμετρικού κλειδιού όπως ο AES. Αλγόριθμοι ασύμμετρου κλειδιού δεν απαιτούν από τα άκρα της επικοινωνίας την ανταλλαγή ενός κοινού κλειδιού αλλά απαιτούν περίπλοκες μαθηματικές τεχνικές που τους καθιστούν δύσχρηστους για μεγάλο όγκο πληροφορίας. Το πρόβλημα επιλύεται με την χρήση του υβριδικού συστήματος.

Η υβριδική κρυπτογράφηση θεωρείται ένας εξαιρετικά ασφαλής τρόπος κρυπτογράφησης των πληροφοριών που ανταλλάσσονται καθώς εκμεταλλεύεται τα θετικά στοιχεία των τύπων κρυπτογράφησης που συνδυάζει[19].

Για της ανάπτυξη τους συστήματος εξουσιοδότησης και την εξασφάλιση της προστασίας των μηνυμάτων που αποστέλλονται μεταξύ αποστολέα και δέκτη. Σε πρώτη φάση υλοποιείται ένας συνδυασμός δύο πολύ διαδεδομένων αλγορίθμων κρυπτογράφησης, του RSA για την ανταλλαγή του AES κλειδιού και έπειτα ο AES για την κρυπτογράφηση και αποκρυπτογράφηση. Έπειτα ο αλγόριθμος RSA αντικαθίσταται με τον αλγόριθμο Diffie Hellman, καταλήγοντας έτσι οι δυο μεριές στο ίδιο AES κλειδί χωρίς όμως αυτό να περνά πάνω από το δίκτυο. Έτσι η κρυπτογράφηση ονομάζεται υβριδική.

## 2.6 Επιτάχυνση υλικού

Αρκετές φορές κατά την διάρκεια σχεδιασμού μιας εφαρμογής προκύπτει η ανάγκη για την ταχύτερη εκτέλεση λειτουργιών και διαδικασιών. Το Hardware Acceleration έρχεται να δώσει λύση σε αυτό το πρόβλημα.

Η «Επιτάχυνση Υλικού» (Hardware Acceleration) είναι η διαδικασία κατά την οποία η εφαρμογή αναθέτει κάποιες από τις αρμοδιότητες της κεντρικής μονάδας επεξεργασίας (CPU) στο υλικό του συστήματος. Οι μονάδες που υλοποιούν επιτάχυνση υλικού είναι ενσωματωμένα κυκλώματα ή μικροεπεξεργαστές που εστιάζουν σε μια λειτουργία τη φορά. Η παραπάνω μέθοδος βελτιώνει την απόδοση της λειτουργίας της εφαρμογής ελευθερώνοντας λίγο τη χρήση της CPU και ταυτόχρονα αναθέτοντας στο εξειδικευμένο υλικό να αναλάβει κάποιες από τις διαδικασίες.

Με τη χρήση του Hardware Acceleration βελτιώνεται η ταχύτητα ολοκλήρωσης λειτουργιών, αυξάνεται η ρυθμαπόδοση και μειώνεται η κατανάλωση ισχύος. Βοηθά σημαντικά σε συστήματα που η διάρκεια λειτουργίας του συστήματος είναι κρίσιμης σημασίας και η διάρκεια της μπαταρίας μπορεί να επιμηκυνθεί.

Το Hardware Acceleration μπορεί να εφαρμοστεί στην κάρτα γραφικών δίνοντας μια «ανάσα» σε browsers μειώνοντας το φόρτο που προσφέρουν οι ιστοσελίδες με τα εντυπωσιακά γραφικά. Μπορεί να εφαρμοστεί για την κωδικοποίηση των video. Συναντάται και κατά τη χρήση της κρυπτογραφίας, αναθέτοντας σε ειδικές μονάδες τους υπολογισμούς τις κρυπτογράφησης και αποκρυπτογράφησης[24].

## 2.7 Βιβλιοθήκες

### 2.7.1 Mbed TLS

Κατά την ανάπτυξη και εξέλιξη των δικτύων υπολογιστών ανταλλάσσονται όλο και μεγαλύτερος όγκος δεδομένων και μηνυμάτων ευαίσθητου περιεχομένου μεταξύ των εφαρμογών τα οποία χρήζουν άμεσης προστασίας. Έτσι προέκυψε η ανάγκη για ιδιωτική και ασφαλή μεταφορά πληροφοριών πάνω από δίκτυα υπολογιστών. Αναπτύχθηκαν τα πρωτόκολλα Secure Sockets Layer (SSL) το 1995 και αργότερα το σύγχρονο Transport Layer Security (TLS) το 2006. Η τελευταία έκδοση του πρωτοκόλλου TLS, έκδοση 1.3, ανακοινώθηκε το 2018 ενώ οι προηγούμενες εκδόσεις δεν χρησιμοποιούνται από το 2021 και μετά[20]. Το πρωτόκολλο TLS στοχεύει στην

πλήρως προστατευμένη, ιδιωτική και ακέραιη μεταφορά των δεδομένων μεταξύ εφαρμογών δύο υπολογιστών πάνω από το δίκτυο.

Το πρωτόκολλο TLS για να παρέχει τις υπηρεσίες ασφαλείας για τις οποίες δημιουργήθηκε, βασίζεται σε πρωτόκολλα κρυπτογραφίας. Σε ένα σύστημα πελάτη-διακομιστή για να μπορέσουν να επικοινωνήσουν τα δύο άκρα χωρίς κίνδυνο υποκλοπής των δεδομένων τους, είναι αναγκαίο οι δύο τους να συμφωνήσουν στην χρήση του πρωτοκόλλου TLS και να εγκαθιδρύσουν μια συνεδρία.

Κατά την διάρκεια αυτής της συνεδρίας, την οποία ξεκινά ο πελάτης γίνεται η συμφωνία διάφορων παραμέτρων για την ομαλή επικοινωνία τους, όπως σχήματα κρυπτογραφίας και συναρτήσεις κατακερματισμού και ανταλλάσσονται τα κλειδιά της κρυπτογράφησης και αποκρυπτογράφησης που θα ακολουθήσει. Η παραπάνω διαδικασία ονομάζεται χειραψία TLS και μαζί με το πρωτόκολλο καταγραφής TLS αποτελούν τα δύο επίπεδα του πρωτοκόλλου TLS.

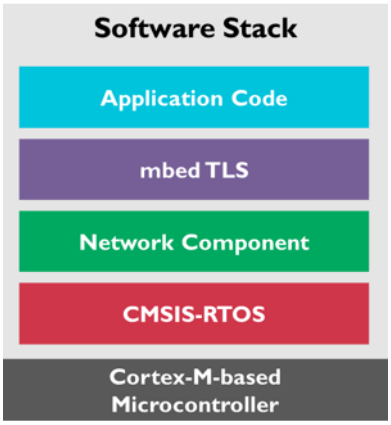
Χρήση του Transport Layer Security πρωτόκολλου γίνεται από ιστοσελίδες και από το ηλεκτρονικό ταχυδρομείο[21].

Ως αποτέλεσμα των παραπάνω σχεδιάστηκε η Mbed TLS, μία βιβλιοθήκη γραμμένη σε γλώσσα C. Η Mbed TLS προσφέρει μια γκάμα βασικών συναρτήσεων για ασφαλή επικοινωνία και πάνω από 6000 unit tests.

Η Mbed TLS σχεδιάστηκε για να είναι όσο το δυνατόν πιο εύκολα προσαρμόσιμη στις ανάγκες του χρήστη. Ο κώδικας της βιβλιοθήκης είναι εύκολα προσβάσιμος και κατανοητός. Σημαντικό είναι το γεγονός ότι μπορούν να χρησιμοποιηθούν συγκεκριμένες συναρτήσεις της βιβλιοθήκης εξατομικευμένα σε κατάλληλα σημεία αποφεύγοντας επιπλέον φόρτο. Ενώ ακόμα παρέχεται η δυνατότητα υλοποίησης συναρτήσεων από τον χρήστη αντί της χρήσης αυτών που προϋπάρχουν. Για την ενεργοποίηση αυτού του αφαιρετικού επιπέδου είναι απαραίτητη ο καθορισμός του αντίστοιχου macro MBEDTLS_XXX_ALT στο αρχείο mbedtls_config.h καθώς και η υλοποίηση στο αντίστοιχο header αρχείο xxx_alt.h.

Στην ασφαλή επικοινωνία πελάτη και διακομιστή με το Mbed TLS εμπλέκονται τα παρακάτω στοιχεία που φαίνονται στην εικόνα που ακολουθεί. Ξεκινώντας από κάτω προς τα πάνω στη στοίβα συναντάμε στο φυσικό επίπεδο έναν μικροελεγκτή βασισμένο σε Cortex-M επεξεργαστή, στην περίπτωση μας το FRDM-K64F και παρέχει τον επεξεργαστή, αποθηκευτικό χώρο, μνήμη και την διεπαφή δικτύου. Έπειτα στο CMSIS-RTOS περιλαμβάνεται το λειτουργικό σύστημα με βασικές παροχές όπως οδηγούς Ethernet , χρονοδρομολόγηση και threads-safety. Στο επίπεδο γίνεται η υλοποίηση των sockets TCP/IP και διεπαφές για την επικοινωνία δικτύου. Με το

επίπεδο Mbed TLS εξασφαλίζεται η ασφαλή επικοινωνία. Τέλος το επίπεδο εφαρμογής χρησιμοποιεί το Mbed TLS για την ασφαλή επικοινωνία της ίδιας της εφαρμογής[22].




*Εικόνα 15: Ιεραρχία Λογισμικού*

Για την χρήση της βιβλιοθήκης σίγουρα απαιτείται το header:

#include "mbed.h"

Ενώ για την χρήση των αλγορίθων RSA και AES απαιτούνται τα headers:

#include "mbedtls/pk.h"

#include "mbedtls/aes.h"

Ενδεικτικά κάποιες από τις συναρτήσεις που παρέχει η βιβλιοθήκη Mbed TLS και χρησιμοποιήθηκαν στην εν λόγω εφαρμογή είναι η mbedtls_ctr_drbg_seed() και mbedtls_ctr_drbg_random() για την δημιουργία μιας τυχαίας συμβολοσειράς. Έπειτα η mbedtls_pk_parse_public_key()για την ανάγνωση ενός RSA public key που έχει ήδη κατασκευαστεί. Ακολουθεί η mbedtls_pk_encrypt()για την κρυπτογράφηση κειμένου με το δημόσιο κλειδί του RSA. Ενώ για τον αλγόριθμο AES απαραίτητο πρώτο βήμα είναι η αρχικοποίηση του AES κλειδιού με την mbedtls_aes_setkey_dec()και έπειτα η κρυπτογράφηση και αποκρυπτογράφηση με τη συνάρτηση mbedtls_aes_crypt_cbc().

Τελικά, η βιβλιοθήκη Mbed TLS της ARM είναι μια ολοκληρωμένη λύση SSL/TLS που διευκόλυνε την ασφαλή επικοινωνία πάνω από δίκτυα και μπορεί να υποστηριχθεί από πολλά λειτουργικά συστήματα όπως Microsoft Windows, Linux, Android, iOS.

### 2.7.2 Βιβλιοθήκη κρυπτογραφικών συναρτήσεων χαμηλού επιπέδου CAU και mmCAU

Αρκετές φορές κατά την διάρκεια σχεδιασμού μιας εφαρμογής προκύπτει η ανάγκη για την ταχύτερη εκτέλεση λειτουργιών και διαδικασιών. Η επιτάχυνση υλικού έρχεται να δώσει λύση σε αυτό το πρόβλημα καθώς στοχεύει στην χρήση του υλικού ενός υπολογιστή ώστε να βελτιωθεί η απόδοση συγκεκριμένων εργασιών [23]. Στα πλαίσια αυτής της ανάγκης σχεδιάστηκε και η βιβλιοθήκη που αναφέρεται παρακάτω.

Η ColdFire/ColdFire+ Crypto Acceleration Unit (CAU) βιβλιοθήκη εμφανίστηκε για πρώτη φορά στις αρχές του 2017 και εστιάζει στην βελτίωση της απόδοσης και της μείωσης της χρησιμοποιούμενης μνήμης σε εφαρμογές που αναφέρονται στην ασφάλεια επικοινωνιών. Είναι ένα σύνολο συναρτήσεων χαμηλού επιπέδου που στοχεύουν στην επιτάχυνση συναρτήσεων κρυπτογραφίας και κατακερματισμού χρησιμοποιώντας το υλικό μιας πλατφόρμας. Η παραπάνω βιβλιοθήκη αναφέρεται στον CAU συνεπεξεργαστή των ColdFire/ColdFire+ συσκευών και Kinetis MCU. Επιπρόσθετα, η CAU μοιράζεται το ίδιο API με την Memory-Mapped Crypto Acceleration Unit (mmCAU) βιβλιοθήκη που αναφέρεται στον συνεπεξεργαστή mmCAU συσκευών Kinetis που διαθέτουν επεξεργαστή ARM Cortex-M4 και συνδέεται με το Private Peripheral Bus (PPB).

Η βιβλιοθήκη παρέχει, σε απλή και εύχρηστη μορφή, μια μεγάλη ποικιλία υλοποιημένων συναρτήσεων κατακερματισμού, κρυπτογράφησης και αποκρυπτογράφησης που χρησιμοποιώντας τον παραπάνω συνεπεξεργαστή των συσκευών στοχεύουν στην επιτάχυνση των αντίστοιχων αλγορίθμων. Υποστηρίζονται αλγόριθμοι AES128, AES192, AES256 σε λειτουργία CBC ,που μελετήθηκαν και στην παρούσα εργασία, αλλά και DES, 3DES, MD5, SHA1, SHA256.

Εστιάζοντας στον αλγόριθμο AES η βιβλιοθήκη παρέχει την συνάρτηση cau_aes_set_key()με την οποία γίνεται η αρχικοποίηση του κλειδιού AES με πρώτο όρισμα έναν δείκτη στο κλειδί και το μέγεθος του κλειδιού, ενώ στο τελευταίο όρισμα αποθηκεύεται το μέγεθος των key schedule που προκύπτουν με βάση την εικόνα που ακολουθεί .

| AES cau_aes_set_key() | |
|---|---|
| **[in]** Block Size (bits) | **[out]** Key Schedule Size (32-bit data values) |
| 128 | 44 |
| 192 | 52 |
| 256 | 60 |

*Εικόνα 16: CAU set up key time παράμετροι*

Στη συνέχεια η συνάρτηση cau_aes_encrypt() υλοποιεί την κρυπτογράφηση. Δέχεται ως πρώτο όρισμα το δείκτη στο κείμενο που πρόκειται να κρυπτογραφηθεί το οποίο χωρίζεται σε κομμάτια των 16 bytes. Έπειτα δείκτη στο key schedule και τον αριθμό των AES γύρων. Δίνει σαν έξοδο το κρυπτογραφημένο κείμενο. Το πλήθος των AES γύρων προκύπτει από την παρακάτω εικόνα.

| AES cau_aes_encrypt()/cau_aes_decrypt() | | |
|---|---|---|
| Block Cipher | **[in]** Key Schedule Size (longwords) | **[in]** Number of AES rounds |
| AES128 | 44 | 10 |
| AES192 | 52 | 12 |
| AES256 | 60 | 14 |

*Εικόνα 17: CAU παράμετροι για κρυπτογράφηση και αποκρυπτογράφηση*

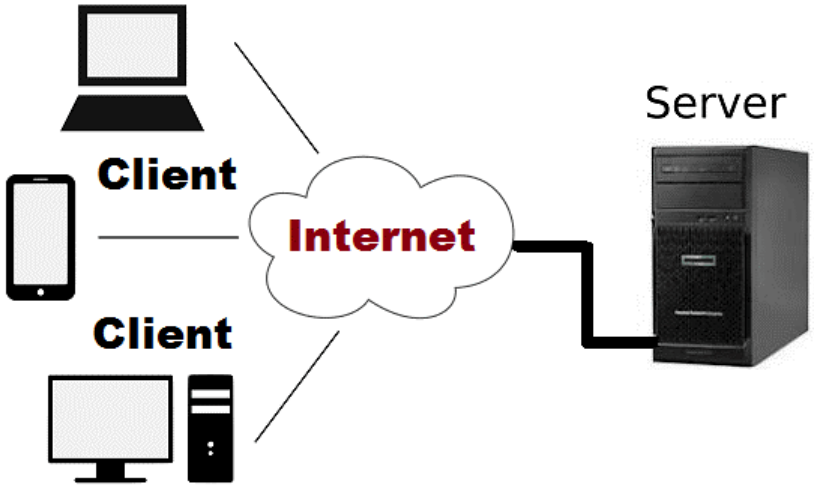
Τέλος με τη συνάρτηση cau_aes_decrypt() αποκρυπτογραφείται ένα κρυπτογραφημένο block. Δέχεται το κρυπτογραφημένο block και σκοπός της είναι να το αποκρυπτογραφήσει και να το επαναφέρει στην αρχική του μορφή.

Συνοψίζοντας, η βιβλιοθήκη με τη χρήση συνεπεξεργαστών που διαθέτουν συγκεκριμένες πλατφόρμας πετυχαίνει τη βέλτιστη απόδοση των κρυπτογραφικών αλγορίθμων και αλγορίθμων κατακερματισμού συγκριτικά με τη χρήση μόνο του λογισμικού. Η κύρια διαφορά μεταξύ την CAU και mmCAU είναι το γεγονός ότι η πρώτη μονάδα είναι προσβάσιμη από τους καταχωρητές του επεξεργαστή ενώ η δεύτερη μονάδα είναι προσβάσιμη σαν εξωτερική περιφερειακή συσκευή του επεξεργαστή και συνδέεται σε αυτόν μέσω του Private Peripheral Bus (PPB). Όσο αφορά τις συναρτήσεις της βιβλιοθήκης όπου καταγράφεται CAU εννοείται και mmCAU, εκτός από κάποιες εξαιρέσεις που σημειώνονται στο documentation.

## 2.8 Τεχνικές Αρχιτεκτονικής λογισμικού

Η Αρχιτεκτονική λογισμικού αναφέρεται στη δομή και στον σχεδιασμό του λογισμικού καθώς και στις διασυνδέσεις των ξεχωριστών στοιχείων που απαρτίζουν το σύστημα. Έτσι, έχουν αναπτυχθεί μεθοδολογίες που βελτιστοποιούν τον σχεδιασμό των συστημάτων ανάλογα με τις ανάγκες της εκάστοτε εφαρμογής. Μία από τις τεχνικές αρχιτεκτονικής λογισμικού είναι και το μοντέλο πελάτη-διακομιστή.

### 2.8.1 Μοντέλο πελάτη-διακομιστή




*Εικόνα 18: Μοντέλο πελάτη-διακομιστή*

Το μοντέλο πελάτη-διακομιστή είναι μια κατανεμημένη δομή λογισμικού όπου ο φόρτος εργασίας διαμοιράζεται στα δύο άκρα του συστήματος που επικοινωνούν μέσω του δικτύου που σχηματίζεται μεταξύ τους.

Στον πραγματικό κόσμο με τον όρο «πελάτη» (client) χαρακτηρίζεται μια ένας άνθρωπος ή οργανισμός που ζητά μια συγκεκριμένη εξυπηρέτηση. Στον ορολογία της πληροφορικής με τον όρο client χαρακτηρίζεται μια συσκευή ή ένας υπολογιστής που δέχεται πληροφορίες ή χρησιμοποιεί υπηρεσίες που παρέχονται από τον εξυπηρετητή. Τέτοιες συσκευές μπορεί να είναι laptops, συσκευές IoT όπως για παράδειγμα η πλατφόρμα FRDM-K64F.

Με τον όρο «διακομιστή» (server) περιγράφεται ένας άνθρωπος ή οργανισμός που παρέχει τις υπηρεσίες του σε όποιον τις ζητήσει. Αυτό συμβαίνει και στον ψηφιακό κόσμο χαρακτηρίζοντας ως servers υπολογιστές που παρέχουν πληροφορίες και υπηρεσίες.

Με τον όρο «Αρχιτεκτονική Πελάτη-Διακομιστή» ή «Μοντέλο Πελάτη-Διακομιστή» περιγράφεται ένα σύστημα που έχει κύριο στόχο την διαχείριση υπηρεσιών και την ανάλυση εργασιών που ζητά ο πελάτης. Με λίγα λόγια, ο πελάτης ζητά κάτι από τον διακομιστή και ο διακομιστής καλείται να ανατρέψει στη βάση δεδομένων που διαθέτει ώστε να ανταποκριθεί στο αίτημα που δέχθηκε.

Τα δύο άκρα υλοποιούν διαφορετικές διεργασίες χρησιμοποιώντας διαφορετικούς πόρους υλικού και λογισμικού και η επικοινωνία τους επιτυγχάνεται μέσω του Διαδικτύου ή δικτύου που σχηματίζεται μεταξύ τους.

Το «Μοντέλο Πελάτη-Διακομιστή» συναντάται στο ηλεκτρονικό ταχυδρομείο για την αποστολή και λήψη των μηνυμάτων μας, στην αποθήκευση αρχείων σε κεντρικούς servers (Cloud) όπως Google Docs.

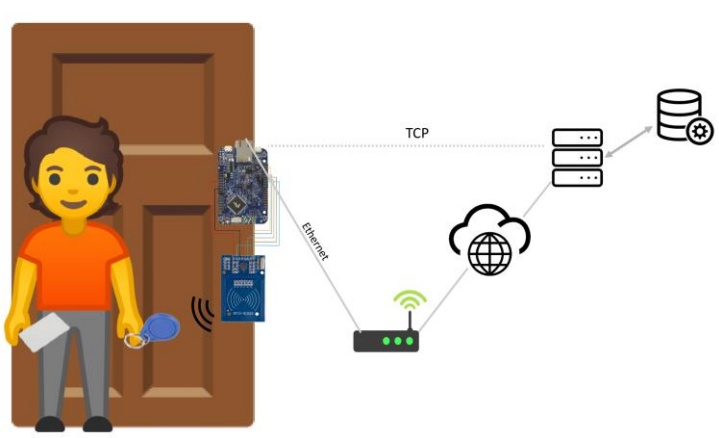
Η παραπάνω αρχιτεκτονική εφαρμόστηκε και στο σύστημα εξουσιοδότησης που αναπτύσσεται στην παρούσα εργασία. Ως πελάτης χαρακτηρίζεται η πλατφόρμα FRDM-K64F και ως διακομιστής η εφαρμογή που επικοινωνεί με τη βάση δεδομένων[25].

# Κεφάλαιο 3. Σύστημα εξουσιοδότησης σε χώρους με κάρτες RFID

Στο κεφάλαιο αυτό περιγράφεται εκτενώς ο αλγόριθμος που αναπτύχθηκε για την λειτουργία του συστήματος εξουσιοδοτημένης πρόσβασης χώρων με κάρτες RFID που πραγματεύεται η εργασία καθώς και η διασύνδεση του υλικού και λογισμικού που το αποτελούν. Αναπτύχθηκαν πρωτόκολλα και σχεδιάστηκαν αλγόριθμοι τόσο για το λογισμικό όσο και για το υλικό που απαρτίζουν αυτό το σύστημα.

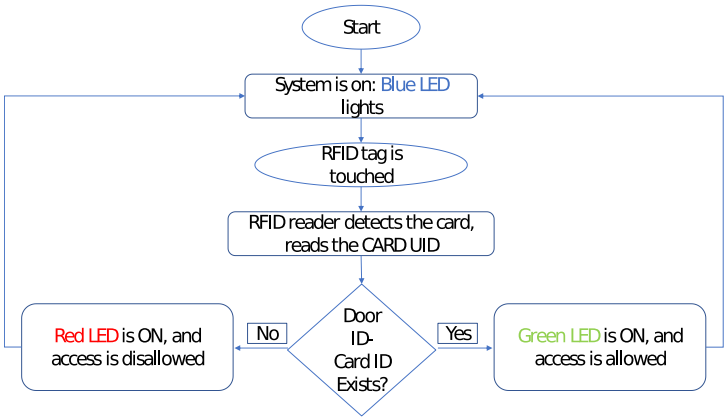
## 3.1 Αρχιτεκτονική του συστήματος

Το σύστημα απαρτίζεται από έναν διακομιστή, τον μικροελεγκτή FRDM-K64F της εταιρείας NXP καθώς και το RFID σύστημα MIFARE RC522 στο οποίο ανήκουν ο RFID αναγνώστης και οι RFID κάρτες. Ο μικροελεγκτής μαζί με τον αναγνώστη του RFID τοποθετούνται στην πόρτα ενώ οι RFID κάρτες δίνονται στους δυνητικούς χρήστες. Ο αναγνώστης του RFID δημιουργώντας ηλεκτρομαγνητικό πεδίο γύρω του, ανιχνεύει το σήμα που εκπέμπεται από την RFID κάρτα που ανήκει στον χρήστη. Το μοναδικό αναγνωριστικό της κάρτας σε συνδυασμό με το αναγνωριστικό του μικροελεγκτή μεταφέρονται στον διακομιστή όπου ελέγχεται η άδεια εισόδου του χρήστη ή μη στο χώρο.




*Εικόνα 19: Αρχιτεκτονική συστήματος αυτόματης πρόσβασης*

# 3.2 Περιγραφή λειτουργίας του συστήματος




*Εικόνα 20: Διάγραμμα ροής λειτουργίας συστήματος*

Το παραπάνω διάγραμμα παρουσιάζει τη ροή της λειτουργίας του αυτόματου συστήματος εξουσιοδότησης με RFID. Το σύστημα ενεργοποιείται, ανάβει το μπλε λαμπάκι και είναι έτοιμο για λειτουργία. Το λαμπάκι εξομοιώνει το servo motor. Όταν ο χρήστης πλησιάσει την κάρτα στον αναγνώστη του RFID, ο αναγνώστης ανιχνεύει το ID της κάρτας και ψάχνει στην βάση δεδομένων για τον συνδυασμό κάρτας ID – πόρτας ID. Εάν ο συνδυασμός υπάρχει το πράσινο λαμπάκι ανάβει και τυπώνεται μήνυμα επιτυχίας. Σε κάθε άλλη περίπτωση ανάβει κόκκινο λαμπάκι και τυπώνεται μήνυμα αποτυχίας.

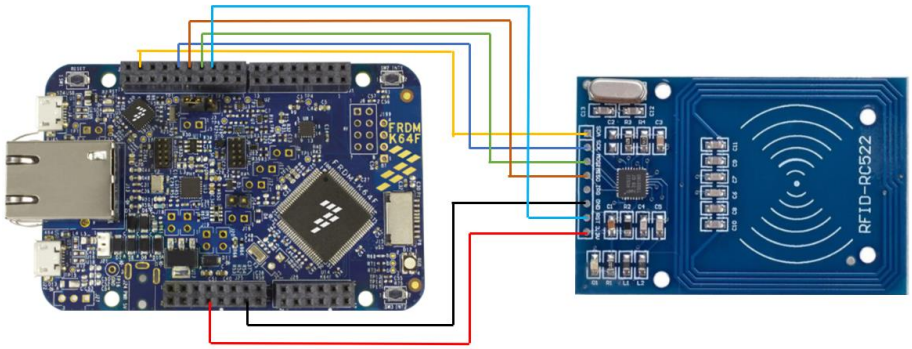
Στην περίπτωση όπου ένας νέος χρήστης θέλει ένα νέο κλειδί προστίθεται το μοναδικό αναγνωριστικό του κλειδιού στη βάση δεδομένων και αντιστοιχείται με το μοναδικό αναγνωριστικό της πόρτας που θα ξεκλειδώνει.

Στην περίπτωση όπου ένας χρήστης δεν έχει το δικαίωμα ξεκλειδώματος της πόρτας ή στην περίπτωση όπου χάσει το κλειδί που του αντιστοιχεί διαγράφεται από τη βάση δεδομένων και απαγορεύεται η πρόσβαση του.

Στην περίπτωση προσθήκης νέας πόρτας στο σύστημα, καταγράφεται ο μοναδικός αριθμός που θα την χαρακτηρίζει στη βάση δεδομένων καθώς και τα κλειδιά που την ξεκλειδώνουν.

## 3.3 Επικοινωνία RFID με FRDM-K64F

Η πλατφόρμα FRDM-K64F επικοινωνεί με το RFID μέσω του πρωτοκόλλου SPI που περιεγράφηκε στην προηγούμενη ενότητα. Για να επιτευχθεί η ανταλλαγή των δεδομένων μεταξύ των συσκευών πραγματοποιήθηκε η ακόλουθη συνδεσμολογία.




*Εικόνα 21: Σχεδιάγραμμα καλωδίωσης*

| PIN | Wiring to FRDM-K64F |
|---|---|
| SPI_MOSI | PTD2 |
| SPI_MISO | PTD3 |
| SPI_SCK | PTD1 |
| SPI_CS | PTE25 |
| MF_RESET | PTD0 |
| GND | GND |
| VCC | 3.3V |

*Εικόνα 22: Καλωδίωση FRDM-K64F με RFID*

## 3.4 Πρωτόκολλο επικοινωνίας μικροελεγκτή και διακομιστή

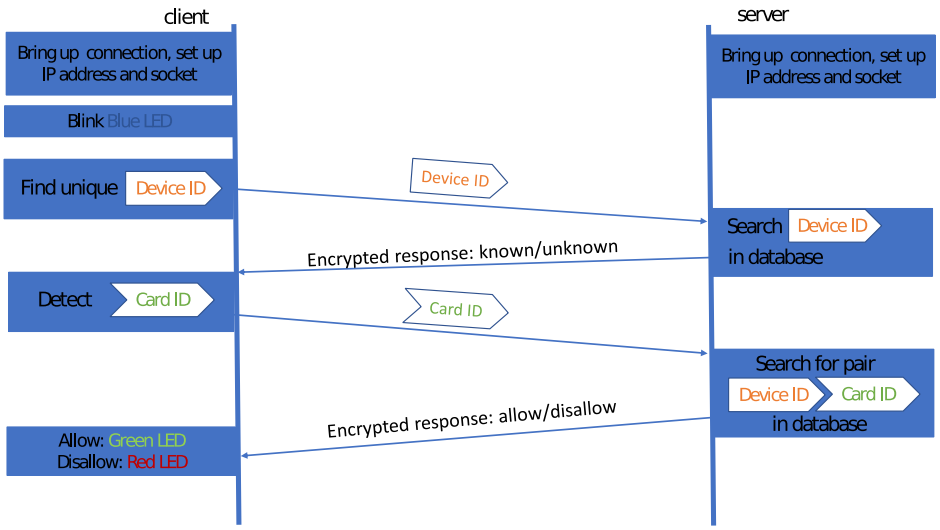
### 3.4.1 Εισαγωγή

Για της ανάγκες του συστήματος εξουσιοδότησης σχεδιάζεται ένα πρωτόκολλο ανταλλαγής μηνυμάτων που έχει στόχο την άρτια και ασφαλή επικοινωνία μεταξύ του πλατφόρμας, που παίζει το ρόλο του πελάτη, και του διακομιστή ο οποίος κατ' επέκταση επικοινωνεί με τη βάση δεδομένων. Το λογισμικό του μικροελεγκτή έχει υλοποιηθεί σε C/C++ ενώ ο server έχει γραφτεί σε Python και συγκεκριμένα στην έκδοση 3.9.

Το πρωτόκολλο που αναπτύχθηκε βασίζεται στο μοντέλο ανάπτυξης λογισμικού πελάτη-διακομιστή που έχει αναλυθεί στο προηγούμενο κεφάλαιο.

### 3.4.2 Ροή βασικού πρωτοκόλλου

Πριν αναλυθεί λεπτομερώς κάθε διαδικασία για την ανάπτυξη του πρωτοκόλλου, θα παρουσιαστούν συνοπτικά η ενέργειες που εκτελούνται από τους κόμβους του δικτύου ώστε να επιτευχθεί ο σκοπός του συστήματος.




*Εικόνα 23: Βασικό πρωτόκολλο*

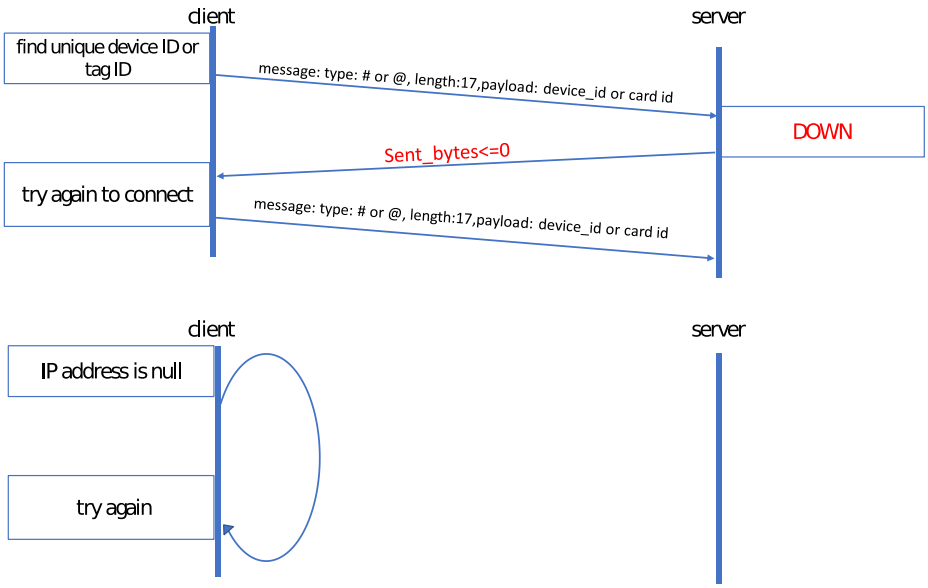
Όπως περιγράφεται από το παραπάνω σχεδιάγραμμα, αρχικό βήμα του αλγορίθμου είναι η εγκαθίδρυση της σύνδεσης και από τις δύο πλευρές ώστε να εξασφαλιστεί η επικοινωνία μεταξύ τους.

Έπειτα ο πελάτης εντοπίζει τον αναγνωριστικό αριθμό του μικροελεγκτή και στέλνει μήνυμα στον server το οποίο δομείται ως εξής: τύπος του μηνύματος, μήκος του μηνύματος και τέλος το ωφέλιμο φορτίο. Ο διακομιστής από τη μεριά του, αφού αναγνωρίσει ότι πρόκειται για μήνυμα που περιέχει το μοναδικό αναγνωριστικό του mcu αναζητά στη βάση δεδομένων και απαντά αναλόγως.

Έπειτα μέσω του RFID reader ο πελάτης εντοπίζει τον μοναδικό αναγνωριστικό αριθμό της ετικέτας που μόλις σαρώθηκε και στέλνει αντίστοιχη δομή μηνύματος με αυτή προηγουμένως. Ο διακομιστής αφού βεβαιωθεί ότι πρόκειται για μήνυμα που αναφέρεται στην ετικέτα αναζητά το συνδυασμό ετικέτας και συσκευής mcu. Αν ο συνδυασμός υπάρχει, τότε ανάβει πράσινο LED αλλιώς κόκκινο LED.

Να σημειωθεί ακόμη πως όσο το σύστημα βρίσκεται σε κατάσταση αναμονής καλώντας την συνάρτηση ανάβει και σβήνει το μπλε LED.

Επιπλέον σε περίπτωση που ο server για οποιονδήποτε λόγω έχει αποσυνδεθεί, η αδυναμία επικοινωνίας εντοπίζεται από τον client και γίνεται επαναποστολή των δεδομένων. Ακόμα εάν αποτύχει η διαδικασία ορισμού διεύθυνσης IP για τον μικροελεγκτή γίνεται ξανά προσπάθεια. Τα δύο αυτά σενάρια περιγράφονται και στο παρακάτω σχήμα.

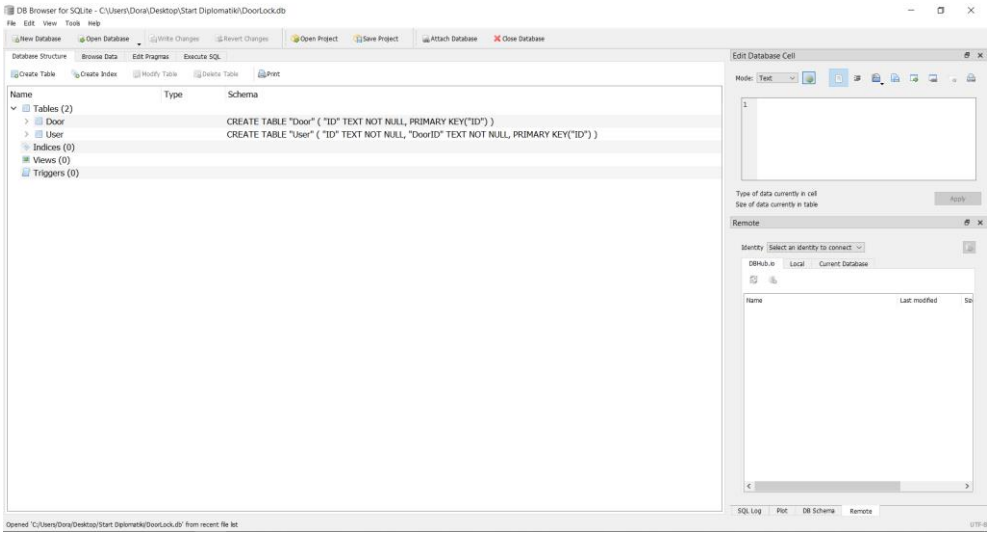



*Εικόνα 24: Σενάρια αποτυχίας εγκαθίδρυσης σύνδεσης*

Ακολουθεί λεπτομερή ανάλυση κάθε μία από τις διαδικασίες που περιεγράφηκαν παραπάνω.

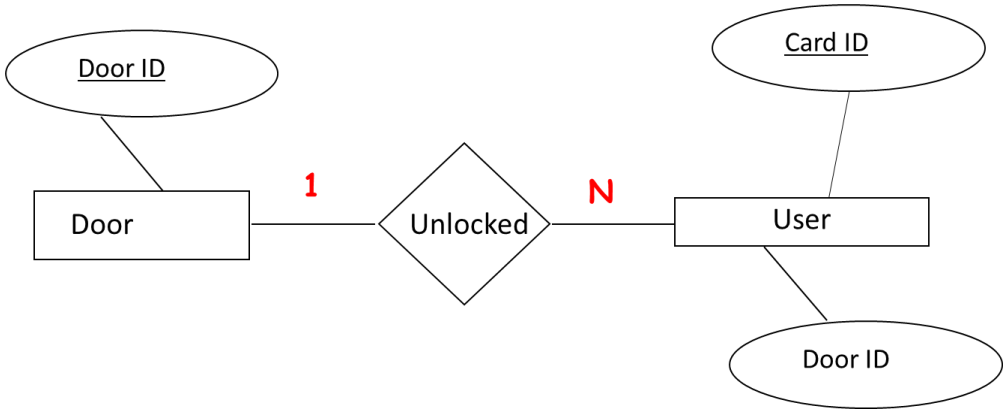
### 3.4.3 Δημιουργία βάσης δεδομένων

Ο σχεδιασμός της βάσης δεδομένων έγινε με τη βοήθεια του open source λογισμικού DB Browser for SQLite. Προσφέρει στον χρήστη ένα απλό και εύχρηστο Interface με το οποίο μπορεί να δημιουργήσει, να σχεδιάσει και να επεξεργαστεί βάσεις δεδομένων με την SQLite.




*Εικόνα 25: DB browser SQL lite*

Η εφαρμογή αντικατοπτρίζει ένα σύστημα πόρτας και χρηστών που μπορούν να την ξεκλειδώσουν. Έτσι υπάρχουν οι πόρτες που αντιστοιχούν στις συσκευές των microcontrollers ενώ τον ρόλο των χρηστών αναλαμβάνουν οι RFID ετικέτες που «ξεκλειδώνουν» την αντίστοιχη πόρτα.




*Εικόνα 26: Σχήμα βάσης δεδομένων*

Από το σχήμα της βάσης δεδομένων γίνεται αντιληπτό ότι μια πόρτα μπορεί να ξεκλειδωθεί από πολλούς χρήστες αλλά κάθε χρήστης μπορεί να ξεκλειδώσει μόνο μία πόρτα. Η πόρτα και ο χρήστης χαρακτηρίζονται μοναδικά με τα αντίστοιχα IDs.

Για τις ανάγκες υλοποίησης της βάσης δεδομένων έχουν δημιουργηθεί δύο πίνακες. Ο πρώτος ονομάζεται Door και περιέχει ένα πεδίο ID το οποίο είναι και κλειδί του πίνακα. Στον πίνακα αυτόν αποθηκεύονται οι αριθμοί που ταυτοποιούν μοναδικά τις συσκευές των microcontrollers.

| ID |
|---|
| Filter |
| ffff0500ffffffff0448454e1c000550 |

*Εικόνα 27: Πίνακας Door της ΒΔ*

Έπειτα δημιουργείται ο πίνακας User που περιέχει δύο πεδία, το ID της πόρτας που ξεκλειδώνει ο κάθε χρήστης αλλά και το ID του κάθε χρήστη.

| ID | DoorID |
|---|---|
| Filter | Filter |
| 3b59a91c | ffff0500ffffffff0448454e1c000550 |
| 3b59a91b | ffff0500ffffffff0448454e1c000550 |

*Εικόνα 28: Πίνακας User της ΒΔ*

Σε περίπτωση εισαγωγής νέου χρήστη προστίθεται στην βάση δεδομένων το ID του κλειδιού που του αντιστοιχεί καθώς και το μοναδικό αναγνωριστικό της πόρτας που ξεκλειδώνει. Σε περίπτωση διαγραφής χρήστη διαγράφεται το κλειδί που του αντιστοιχούσε.

### 3.4.4 Εύρεση μοναδικού κλειδιού της πλατφόρμας.

Αρχικό βήμα του αλγορίθμου είναι η εύρεση του αριθμού που προσδιορίζει μοναδικά την κάθε συσκευή του μικροελεγτκή, δηλαδή το device_id. Με τον αριθμό αυτό ταυτοποιείται κάθε μικροελεγκτής που είναι μέρος του συστήματος και αποθηκεύεται στη βάση δεδομένων με την οποία επικοινωνεί ο διακομιστής. Σύμφωνα με το εγχειρίδιο της πλατφόρμας στις τέσσερεις συγκεκριμένες θέσεις μνήμης που φαίνονται στην εικόνα αναπαρίσταται ένας 128-bits μοναδικός αριθμός για τον μικροελεγκτή.

| | | | | | |
|---|---|---|---|---|---|
| 4004_8054 | Unique Identification Register High (SIM_UIDH) | 32 | R | See section | 12.2.19/327 |
| 4004_8058 | Unique Identification Register Mid-High (SIM_UIDMH) | 32 | R | See section | 12.2.20/327 |
| 4004_805C | Unique Identification Register Mid Low (SIM_UIDML) | 32 | R | See section | 12.2.21/328 |
| 4004_8060 | Unique Identification Register Low (SIM_UIDL) | 32 | R | See section | 12.2.22/328 |

*Εικόνα 29: Διευθύνσεις μνήμης αναγνωριστικού αριθμού MCU*

Ο μοναδικός αυτός αριθμός εντοπίζεται στην συνάρτηση findUniqueDeviceId()και αποθηκεύεται στον πίνακα που δέχεται ως όρισμα η συνάρτηση. Στον πίνακα IdentificationRegistersAddresses[] αποθηκεύονται οι διευθύνσεις στις οποίες βρίσκεται οι χρήσιμη πληροφορία. Έπειτα με τη βοήθεια του δείκτη pointer μεταφέρονται τα 4 bytes της κάθε διεύθυνσης στον πίνακα device_id μεγέθους 16 bytes που κρατά την ολοκληρωμένη πληροφορία. Μπορείτε να βρείτε τον κώδικα στο Παράρτημα Α.

### 3.4.5 Εύρεση του αριθμού της RFID ετικέτας

Για την επικοινωνία του microcontroller και του RFID reader έγινε χρήση της Arduino βιβλιοθήκης MFRC522 που βρίσκεται στο GitHub η οποία είναι συμβατή τόσο με το mbed OS όσο και με το FRDM-K64F. Κατά την υλοποίηση της εφαρμογής χρησιμοποιήθηκαν τα αρχεία MFRC522.h και MFRC522.cpp της βιβλιοθήκης.

Στη συνάρτηση `findTagUniqueIDAndPost()` και με τη βοήθεια των συναρτήσεων που παρέχονται από τη βιβλιοθήκη MFRC522 αποθηκεύεται ο αριθμός της ετικέτας στον πίνακα `card_id` μεγέθους 5 bytes. Αρχικά δημιουργείται αντικείμενο της κλάσης και ορίζονται τα pins για την λειτουργία του πρωτοκόλλου SPI με το οποίο επικοινωνεί ο αισθητήρας και ο microcontroller.

```
MFRC522 RfChip(PTD2, PTD3, PTD1, PTE25, PTD0);
```

Η διαδικασία ξεκινά με την αρχικοποίηση του `RC522 chip` με την κλήση της `PCD_Init()`. Έπειτα με τη συνάρτηση `PICC_IsNewCardPresent()` ανιχνεύονται νέες συσκευές στο ηλεκτρομαγνητικό πεδίο γύρω από τον RFID αναγνώστη ενώ με την συνάρτηση `PICC_ReadCardSerial()` διαβάζεται ο αριθμός μίας από τις κάρτες που έχουν ανιχνευτεί.

Τέλος, αποθηκεύεται στον `card_id` το ID της ετικέτας αφού πρώτα έχει μετατραπεί σε bytes με τη χρήση της `RfChip.uid.uidByte`.

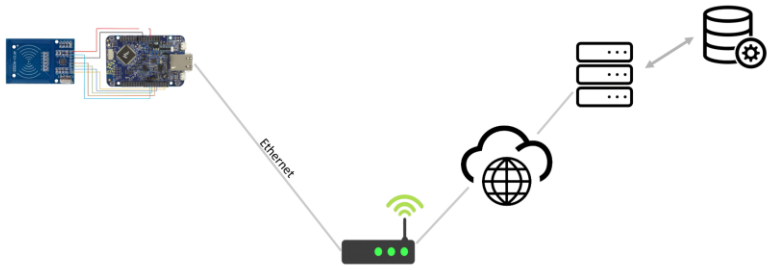
Όπως υποδηλώνεται και από το όνομα της συνάρτησης `findTagUniqueIDAndPost()`στην αρμοδιότητά της είναι και η αποστολή του

μηνύματος στον διακομιστή με κλήσης της συνάρτησης `sendMessageToServer()` η λειτουργία της οποίας θα εξηγηθεί παρακάτω.

Οι κάρτες το RFID συστήματος δεν έχουν τη δυνατότητα αλλαγής του αναγνωριστικού τους γεγονός που μπορεί να προκαλέσει προβλήματα όσο αφορά την ασφάλεια του συστήματος. Μπορείτε να βρείτε τον αναφερόμενο κώδικα στο Παράρτημα Α.

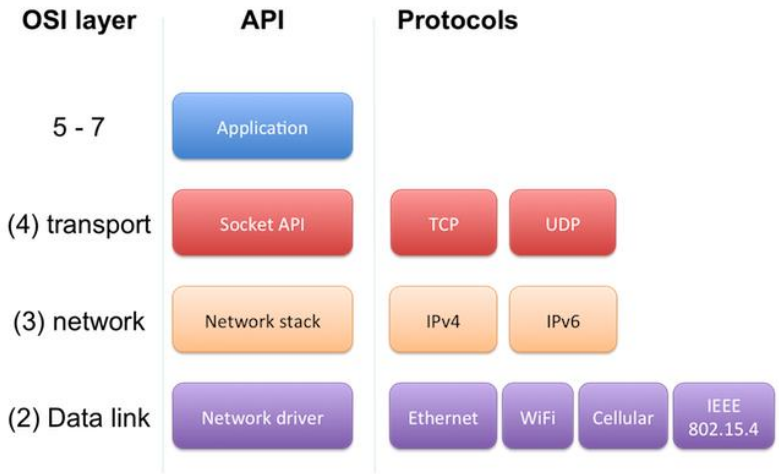
### 3.4.6 Εγκαθίδρυση σύνδεσης μεταξύ πελάτη-διακομιστή.

Απαραίτητη προϋπόθεση για να μπορέσει να επικοινωνήσει πελάτης και διακομιστής είναι να βρίσκονται στο ίδιο δίκτυο. Για να συμβεί αυτό ο microcontroller συνδέεται μέσω του πρωτοκόλλου Ethernet στο router του δικτύου και έπειτα ο server χρησιμοποιεί το πρωτόκολλο IEEE 802.11 (Wi-Fi) για την σύνδεση του στο ίδιο δίκτυο. Η τοπολογία του δικτύου φαίνεται στο παρακάτω σχήμα.




*Εικόνα 30: Τοπολογία συστήματος*

#### 3.4.6.1 Από την μεριά του πελάτη




*Εικόνα 31: IP Networking*

Για να επιτευχθεί η σύνδεση του μικροελεγκτή στο δίκτυο μέσω πρωτοκόλλου Ethernet, γίνεται χρήση του API `EthernetInterface` που παρέχεται από το mbed OS.

Αρχικά δημιουργείται αντικείμενο `net` της κλάσης `EthernetInterface`. Έπειτα στη συνάρτηση `bringUpEthernetConnection()` ξεκινά η σύνδεση στο δίκτυο καλώντας την `connect()`. Εν συνεχεία, δημιουργείται αντικείμενο της κλάσης `SocketAddress` και καθορίζεται το μοναδικό ζεύγος IP διεύθυνσης και πόρτας που θα χαρακτηρίσουν τον πελάτη ως κόμβο του δικτύου. Με την κλήση της συνάρτησης `get_ip_address()` μετατρέπεται η διεύθυνση που δόθηκε στον microcontroller από το δυαδικό σύστημα σε string ώστε να γίνει ευανάγνωστη.

Για την μεταφορά των δεδομένων είναι απαραίτητη η δημιουργία TCP socket. Αυτό επιτυγχάνεται με το `TCPSocket` API που προσφέρει το mbed OS. Χρησιμοποιείται το πρωτόκολλο TCP που παρέχει σταθερή σύνδεση και εγγυάται αξιοπιστία στη μεταφορά των πακέτων ενώ παράλληλα εξασφαλίζει τη σωστή σειρά ανάγνωσης των πακέτων από τον παραλήπτη.

Δημιουργείται αντικείμενο της κλάσης `TCPSocket` και με την κλήση της `open()` ανοίγει το socket στο Ethernet Interface που δημιουργήθηκε προηγουμένως. Καλείται η συνάρτηση `gethostbyname()` η οποία δέχεται ως πρώτο όρισμα την IP διεύθυνση που έχει δοθεί από το router στον διακομιστή, και ως δεύτερο όρισμα το αντικείμενο `SocketAddress` που έχει ήδη δημιουργηθεί. Η πόρτα επικοινωνίας είναι η 8080 που τέθηκε με την συνάρτηση `set_port()`.

Τελικά καλώντας την `connect()` εγκαθιδρύεται η απομακρυσμένη TCP σύνδεση έχοντας ως όρισμα τη IP διεύθυνση του μικροελεγκτή.

Σε περίπτωση που προκύψει σφάλμα η συνάρτηση επιστρέφει 1 και γίνεται προσπάθεια εγκαθίδρυσης της σύνδεσης μέχρις ότου να αποκατασταθεί. Ο κώδικας της ενότητας βρίσκεται στο Παράρτημα Α.

#### 3.4.6.2 Από την μεριά του διακομιστή

Όπως και στον client έτσι και στον server είναι απαραίτητη η δημιουργία socket. Για την υλοποίηση των sockets προσφέρεται από την Python το `Socket` API, το οποίο παρέχει πληθώρα συναρτήσεων που διευκολύνουν αυτή τη διαδικασία.

Η διαδικασία ξεκινά με την δημιουργία αντικειμένου socket με τη συνάρτηση `socket.socket()`. Η συνάρτηση λαμβάνει δύο ορίσματα, το `socket.AF_INET`

το οποίο ορίζει την οικογένεια διευθύνσεων και συγκεκριμένα IPv4 ενώ το `socket.SOCK_STREAM` καθορίζει τον τύπο του socket, δηλαδή το πρωτοκόλλου που πρόκειται να χρησιμοποιηθεί για τη μεταφορά των μηνυμάτων στο δίκτυο. Στην παρούσα περίπτωση θα χρησιμοποιηθεί το πρωτόκολλο TCP όπως έχει προαναφερθεί.

Στη συνέχεια, με τη συνάρτηση `bind()` και με τα ορίσματα τα οποία δέχεται αυτή, ανατίθεται αντίστοιχα στο αντικείμενο του socket που δημιουργήθηκε ένας συνδυασμό διεύθυνσης και αριθμού πόρτας, στην συγκεκριμένη περίπτωση `"0.0.0.0"` και `8080` αντίστοιχα.

Αμέσως μετά, ο server είναι έτοιμος να δεχθεί συνδέσεις και αυτό μπορεί να επιτευχθεί με την `listen()` η οποία καθορίζει τον αριθμό των νέων συνδέσεων που μπορεί να διαχειριστεί ο διακομιστής.

```
ServerSideSocket=socket.socket()
ServerSideSocket.bind(("0.0.0.0", PORT))
ServerSideSocket.listen(5)
```

Για τις απαιτήσεις του συστήματος εξουσιοδότησης μία σύνδεση ήταν αρκετή αλλά σε περίπτωση επεκτασιμότητας υπάρχει η δυνατότητα διαχείρισης περισσότερων πελατών.

Για κάθε νέα σύνδεση καλείται η συνάρτηση `accept()`.

Με τη συνάρτηση `accept()` δημιουργείται ένα νέο socket για τον κάθε πελάτη. Κάθε νέα σύνδεση τοποθετείται σε διαφορετικά threads καλώντας την συνάρτηση `start_new_threads()`. Έτσι επιτυγχάνεται ένας αποδοτικός τρόπος για την ταυτόχρονη λήψη μηνυμάτων από πολλούς πελάτες. Ο διακομιστής λαμβάνει διαρκώς νέα δεδομένα από τον πελάτη με τη συνάρτηση `recv()`. Σε περίπτωση που ο

διακομιστής θελήσει να στείλει απάντηση στον πελάτη καλείται η συνάρτηση `send()`, θα δοθούν περισσότερες πληροφορίες παρακάτω. Ο κώδικας της ενότητας βρίσκεται στο Παράρτημα Α.

## 3.4.7 Αποστολή και λήψη μηνυμάτων

### 3.4.7.1 Από την μεριά του πελάτη

Αφού ολοκληρωθεί η εγκαθίδρυση της επικοινωνίας μεταξύ των δύο άκρων του δικτύου ήρθε η στιγμή για την αποστολή των βασικών μηνυμάτων. Απαραίτητη προϋπόθεση στην ανάπτυξη του πρωτοκόλλου είναι η δομημένη επικοινωνία μεταξύ πελάτη και διακομιστή. Για το λόγω αυτό κάθε μήνυμα ακολουθεί το εξής πρότυπο:

```
struct message
{
  char type;
  int length;
  char *payload;
}
```

Ο πρώτος χαρακτήρας του μηνύματος υποδηλώνει τον τύπο του. Υπάρχουν τέσσερεις τύποι μηνυμάτων ώστε να μπορέσει ο διακομιστής να αντιληφθεί αυτό που ζητείται από τον πελάτη και να ανταποκριθεί αναλόγως. Οι τύποι τον μηνυμάτων είναι οι ακόλουθοι :

- Ο χαρακτήρας «#» αναφέρεται σε μήνυμα που περιέχει το μοναδικό αναγνωριστικό του μικροελεγκτή
- Ο χαρακτήρας «@»προσδιορίζει το μήνυμα που περιέχει τον μοναδικό αριθμό της ετικέτας του RFID
- Ο χαρακτήρας «!» καθορίζει το μήνυμα αιτήματος του πελάτη για το RSA δημόσιο κλειδί
- Ο χαρακτήρας «^» χαρακτηρίζει το κρυπτογραφημένο aes key για το οποίο επίσης θα δοθούν εξηγήσεις αργότερα
- Ο χαρακτήρας «P» χαρακτηρίζει το αίτημα του πελάτη προς τον διακομιστή για την αποστολή της παραμέτρου P του αλγορίθμου Diffie Hellman

- Ο χαρακτήρας «G» χαρακτηρίζει το αίτημα του πελάτη προς τον διακομιστή για την αποστολή της παραμέτρου G του αλγορίθμου Diffie Hellman
- Ο χαρακτήρας «&» σημαίνει ότι ο πελάτης στέλνει στον διακομιστή μήνυμα που περιέχει συγκεκριμένη παράμετρο για τον αλγόριθμο Diffie Hellman.
- Ο χαρακτήρας «*» που αποστέλλεται από τον πελάτη προς το διακομιστή σημαίνει αίτημα για ενημέρωση σχετικά με την μέθοδο ανταλλαγής κλειδιών που έχει επιλέξει ο χρήστης

Έπειτα καθορίζεται το μήκος του μηνύματος και τέλος ακολουθεί η ωφέλιμη πληροφορία που πρέπει να φτάσει αναλλοίωτη στον διακομιστή.

Η διαδικασία της αποστολής μηνυμάτων γίνεται στη συνάρτηση `sendMessageToServer()` και δέχεται σαν όρισμα το socket που χρησιμοποιείται στην επικοινωνία αλλά και μια δομή μηνύματος. Με τη συνάρτηση `send()` της βιβλιοθήκης `TCPSocket` γίνεται η αποστολή του τύπου, του ωφέλιμου φορτίου (συμπεριλαμβάνεται και ο τύπος) και του μήκους των δεδομένων. Η συνάρτηση επιστρέφει ως ακέραιο το αριθμό των bytes που κατάφερε να λάβει ο διακομιστής.

```
int sent_bytes = (*socket).send(msg.payload, msg.length);
```

Σε περίπτωση που αυτός ο αριθμός είναι αρνητικός υποδηλώνεται ότι ο διακομιστής για οποιονδήποτε λόγω είναι αποσυνδεδεμένος και είναι ανάγκη να γίνει ξανά προσπάθεια από τον microcontroller για αποστολή των δεδομένων. Η παραπάνω διαδικασία ελέγχεται στη συνάρτηση `checkIfServerIsDown()` ενώ το LED του μικροελεγκτή γίνεται πορτοκαλί .

Η λήψη των δεδομένων που στέλνονται από τον διακομιστή γίνεται στη συνάρτηση `receiveResponseFromServer()` . Με την κλήση της `recv()` γίνεται η λήψη των δεδομένων και γεμίζεται ο `rbuffer`.

```
int rcount = (*socket).recv(rbuffer, sizeof rbuffer);
```

Στη συνάρτηση `decryptMessage()` μετατρέπεται η απάντηση του διακομιστή από bytes σε string. Αν η απάντηση αφορά την εύρεση του ID του mcu στη βάση δεδομένων, στο monitor τυπώνεται θετική ή αρνητική απάντηση αντίστοιχα. Εάν η απάντηση αφορά συνδυασμό ID microcontroller και ID ετικέτας τότε στο monitor τυπώνεται success ή fail και ανάβει πράσινο ή κόκκινο LED αντίστοιχα. Το μήνυμα επιτυχίας υποδηλώνει ότι ο συνδυασμός βρέθηκε στη βάση

δεδομένων και δίνεται πρόσβαση στο χρήστη. Ο κώδικας της ενότητας βρίσκεται στο Παράρτημα Α.

#### 3.4.7.2 Από την μεριά του διακομιστή

Η λήψη και η αποστολή των δεδομένων από τη μεριά του διακομιστή γίνεται με τις συναρτήσεις `send()` και `recv()` που προσφέρονται από την βιβλιοθήκη `Socket` της Python.

Η `recv()` περιμένει μέχρι να διαβάσει κάποιο byte δεδομένων από το socket που έχει ανοιχτεί και η πληροφορία αποθηκεύεται στο buffer `data`. Μπορεί να διαβάζει έως και 4096 bytes, όσο είναι δηλαδή και το buffersize που έχουμε ορίσει.

Μετέπειτα η ενέργεια που θα κάνει ο διακομιστής εξαρτάται από τον τύπου του μηνύματος που μόλις έχει λάβει. Για να μπορέσει να διακρίνει τον τύπο του μηνύματος απομονώνει και ελέγχει τα δύο πρώτα bytes του πίνακα `data` μετατρέποντας τα bytes σε UTF-8 format με τη βοήθεια της `decode()`.

Στην πρώτη περίπτωση βλέποντας το σύμβολο "#", ο διακομιστής αντιλαμβάνεται ότι μόλις έλαβε ID του microcontroller και καλείται η `search_for_device_id_in_database()`. Όπως υποδηλώνει και το όνομά της αναζητά στη βάση δεδομένων το ID της συσκευής του mcu. Για την αναζήτηση στη βάση γράφτηκε το εξής query :

```
"SELECT ID FROM `Door` WHERE ID = ?", [device_id]
```

Αν βρεθεί το μοναδικό αναγνωριστικό στη βάση επιστρέφεται 0.

Αν πρόκειται για τον αριθμό της RFID ετικέτας με τη συνάρτηση `decode()` εντοπίζει τον χαρακτήρα "@" στα πρώτα δύο bytes του πίνακα data και καλεί τη συνάρτηση `search_for_pair_in_database()`. Για τον εντοπισμό του ζητούμενου ζεύγους γράφτηκε το query:

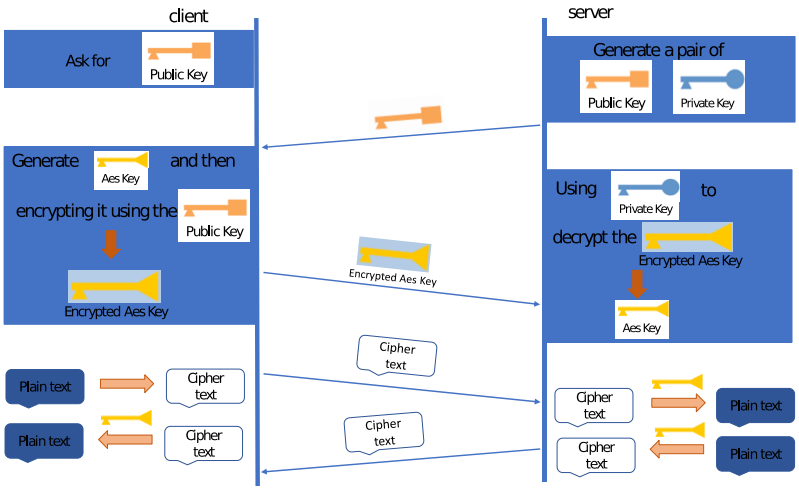
```
"SELECT DoorID FROM `User` WHERE ID = ?", [card_id]
```

Ελέγχει την ομοιότητα του `DoorID` και του `device_id`. Αν υπάρχει το ζεύγος τότε επιστρέφεται 0 αλλιώς 1.

Σε κάθε περίπτωση η απάντηση προς το πελάτη στέλνεται με τη συνάρτηση `send()`έχοντας ως όρισμα τα δεδομένα που πρόκειται να σταλθούν. Ο πελάτης θα λάβει την απάντηση και είναι υπεύθυνος να τα διαχειριστεί κατάλληλα.

## 3.5 Υβριδική κρυπτογραφία

### 3.5.1 Ροή αλγορίθμου υβριδικής κρυπτογραφίας με RSA και AES




*Εικόνα 32: Ροή αλγορίθμου υβριδικής κρυπτογραφίας με RSA και AES*

Όπως φαίνεται από την παραπάνω εικόνα, ο πελάτης ζητά το δημόσιο κλειδί(public key) που έχει ήδη δημιουργήσει ο διακομιστής ώστε να ξεκινήσει η διαδικασία της κρυπτογράφησης. Έπειτα ο πελάτης παράγει ένα συμμετρικό aes key και το κρυπτογραφεί με τη βοήθεια του δημόσιου κλειδιού που μόλις έλαβε. Το κρυπτογραφημένο aes key αποστέλεται στον διακομιστή ο οποίος με την σειρά του το αποκρυπτογραφεί με βάση το ιδιωτικό κλειδί του αλγορίθμου RSA που έχει παράξει προηγουμένως. Επομένως και οι δύο μεριές της επικοινωνίας διαθέτουν τώρα το συμμετρικό aes κλειδί. Κάθε μήνυμα που μεταφέρεται στο δίκτυο κρυπτογραφείται με το aes key από την μεριά του αποστολέα (είτε είναι ο πελάτης είτε ο διακομιστής) και αποκρυπτογραφείται με από την μεριά του παραλήπτη. Έτσι το αρχικό μήνυμα ενώ ανταλλάσσεται αλλοιωμένο στο δίκτυο οι δύο κόμβοι μπορούν να το κατανοήσουν.

Ακολουθεί λεπτομερή ανάλυση κάθε μία από τις διαδικασίες που περιεγράφηκαν παραπάνω.

### 3.5.2 Δημιουργία και ανταλλαγή κλειδιών κρυπτογραφίας με RSA.

#### 3.5.2.1 Από την μεριά του διακομιστή

Σύμφωνα με την διάγραμμα της παραπάνω ενότητας 3.4.1 αρμοδιότητα του διακομιστή είναι η δημιουργία του δημόσιου και ιδιωτικού κλειδιού που θα χρησιμοποιηθούν στον αλγόριθμο RSA. Η Python με τη βοήθεια της βιβλιοθήκης Crypto παρέχει μια μεγάλη γκάμα συναρτήσεων και αλγορίθμων κρυπτογραφίας. Συγκεκριμένα το module `Crypto.PublicKey.RSA` προσφέρει έτοιμες συναρτήσεις που διευκολύνουν την υλοποίηση του αλγορίθμου RSA.

Στη `generate_rsa_keys()` με την `RSA.generate(1024)` δημιουργείται ένα αντικείμενο RSA key το οποίο περιλαμβάνει το ζεύγος ιδιωτικού και δημόσιου κλειδιού. Το μέγεθος του κλειδιού είναι 1024 bits.

```
new_key = RSA.generate(1024)
```

Έπειτα γίνεται εξαγωγή των κλειδιών επιλέγοντας PEM format όπως έχει προκαθοριστεί.

```
private_key = new_key.exportKey("PEM")
public_key = new_key.publickey().exportKey("PEM")
```

Η μορφή PEM (Privacy-enhanced Electronic Mail) χρησιμοποιείται για την αποθήκευση κρυπτογραφημένων δεδομένων και το σώμα της πληροφορίας κωδικοποιείται χρησιμοποιώντας Base64. Η μορφή αυτή μπορεί εύκολα να αναγνωριστεί αφού υπάρχει κεφαλίδα και υποσέλιδο που αποτελείται από παύλες και τις λέξεις BEGIN και END δηλώνοντας την αρχή και το τέλος αντίστοιχα.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAk9zv6gtOlFLueEhjN4Wc
unNIqVyMQsY6kt0Rheau/lBTMI4ws1x6032eiu78YhGGaOevzo16XdfSt+0sLBa5
YB1KVwUXs9hf3bIMvKb7dLmsy+XlKhmBO/bqjekYV9CYjKaGOqrH5TT3nmQSTse7
PvmJ8kL0v5mGTB7bHxr2PJYit5U93zbc2bRrBdLmNyZQYJMakO73ZeqIS9xyk23+
54kVfHakGRsg8Tn5ARHYn+ujJD3Mi30NxdrPArSq7xncDw4rY1vXMZR/JLb/YMQA
gxbQFp68vIswqYEDTSEhwyP0mfEX/NBtKpdwbD8xeunF+QUlacJjT9JEhPuA4IH8
OwIDAQAB
-----END PUBLIC KEY-----
```

*Εικόνα 33: RSA Δημόσιο κλειδί σε PEM format*

Τέλος τα κλειδιά γράφονται σε αρχεία.

Ο κώδικας της ενότητας βρίσκεται στο Παράρτημα Α.

#### 3.5.2.2 Από την μεριά του πελάτη

Ο πελάτης από τη δική του μεριά είναι υπεύθυνος για την δημιουργία του κλειδιού του αλγορίθμου AES. Για την δημιουργία του κλειδιού προσφέρεται το Mbed TLS API από την εταιρία ARM διευκολύνοντας την διαδικασία.

Το κλειδί AES είναι μια συμβολοσειρά τυχαίων αριθμών. Ένα τέτοιο κλειδί θεωρείται ισχυρό αφού βασίζεται στο απρόβλεπτο της τυχαιότητας. Για το σκοπό αυτό έγινε χρήση του module `ctr_drbg` αλλά και της εντροπίας που προσφέρει το Mbed TLS.

Αφού συμπεριλήφθηκαν στον κώδικα τα απαραίτητα headers.

```
#include "mbedtls/entropy.h"
#include "mbedtls/ctr_drbg.h"
```

Και έγιναν οι απαραίτητες αρχικοποιήσεις των μεταβλητών.

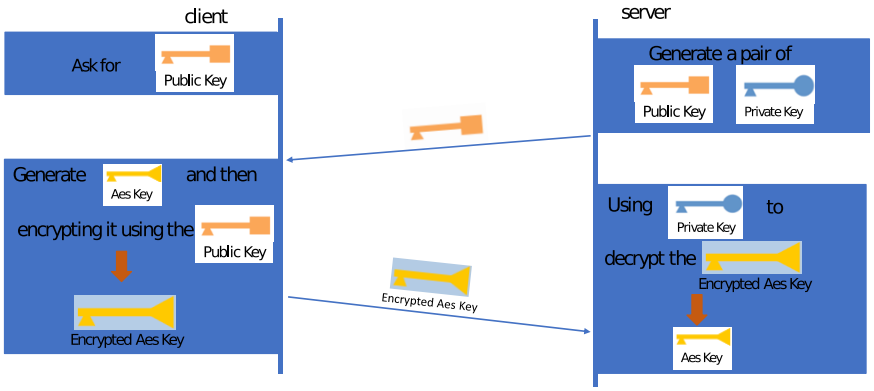
```
mbedtls_ctr_drbg_context ctr_drbg;
mbedtls_entropy_context entropy;

char *pers = "aes generate key";
int ret;
```

Με το string pers προστίθεται επιπλέον τυχαίοτητα στην παραγωγή τυχαίων συμβόλων.

Τελικά στην μεταβλητή `aes_key` μεγέθους 32 bytes αποθηκεύεται μία σειρά τυχαίων συμβόλων που αποτελούν το κλειδί του AES αλγορίθμου. Ο κώδικας της ενότητας βρίσκεται στο Παράρτημα Α και βασίζεται στο documentation του Mbed TLS[7].

#### 3.5.2.3 Ανταλλαγή δημόσιου κλειδιού με RSA

Αφού η κάθε μεριά έχει δημιουργήσει τα κλειδιά των αλγορίθμων που θα βοηθήσουν στην υλοποίηση της υβριδικής κρυπτογραφίας, είναι η στιγμή για την ανταλλαγή αυτών των κλειδιών που απαιτούνται για την ανάγκες της κρυπτογράφησης και αποκρυπτογράφησης.




*Εικόνα 34: Ανταλλαγή κλειδιών με RSA*

Ο πελάτης έχει δημιουργήσει το AES κλειδί το οποίο πρέπει να σταλθεί στον διακομιστή για την κρυπτογράφηση και αποκρυπτογράφηση των μηνυμάτων. Όμως κάθε μήνυμα πάνω από το δίκτυο οφείλει να είναι αλλοιωμένο προς αποφυγή κλοπής της πληροφορίας. Για να συμβεί αυτό θα υλοποιηθεί RSA κρυπτογράφηση.

Στην `askForPublicKey()` ζητά το public key που έχει προηγουμένως δημιουργήσει ο διακομιστής. Δημιουργεί ένα νέο μήνυμα όπως αναλύθηκε σε προηγούμενο κεφάλαιο ο τύπος του οποίου δηλώνεται με το σύμβολο "!".

```
char hello_msg[] = "!";
```

Και αποστέλλει το αίτημα του με τη συνάρτηση `sendMessageToServer()`.

Ο διακομιστής απομονώνοντας το πρώτο byte του μηνύματος αντιλαμβάνεται ότι πρόκειται για αίτημα δημόσιου κλειδιού και απαντά στέλνοντας το public key.

```
if (data[0:1].decode("utf-8") == "!"):
    sockfd.send(public_key)
```

Ο πελάτης λαμβάνει το δημόσιο κλειδί και συνεχίζει κρυπτογραφώντας το AES key ώστε να σταλθεί με ασφάλεια στον διακομιστή. Για να κρυπτογραφηθεί το AES κλειδί ακολουθείται το documentation του Mbed TLS[8] και οι συναρτήσεις που προσφέρει αυτό. Αρχικά προστίθεται στον κώδικα το ανάλογο header.

```
#include "mbedtls/pk.h"
```

Έπειτα ακολουθεί η αρχικοποίηση απαραίτητων μεταβλητών και γίνεται η ανάγνωση του rsa public key που έχει ληφθεί νωρίτερα.

Τελικά κρυπτογραφείται το AES key με την `mbedtls_pk_encrypt()` και το κρυπτογραφημένο αποτέλεσμα αποθηκεύεται στην μεταβλητή `buf`.

Για να σταλθεί το κρυπτογραφημένο κλειδί στον διακομιστή τροποποιείται για να ανταποκρίνεται στη δομή του μηνύματος που έχει καθοριστεί από το πρωτόκολλο προσθέτοντας το σύμβολο “^” και διευκολύνοντας έτσι την αναγνώριση του από τον διακομιστή.

Αφού λάβει ο διακομιστής το κρυπτογραφημένο aes κλειδί ξεκινά η αποκρυπτογράφησης. Η διαδικασία αρχίζει διαβάζοντας το ιδιωτικό κλειδί RSA που έχει δημιουργηθεί νωρίτερα και έχει αποθηκευτεί στο αντίστοιχο αρχείο.

```
key = RSA.import_key(open('private_key.pem').read())
```

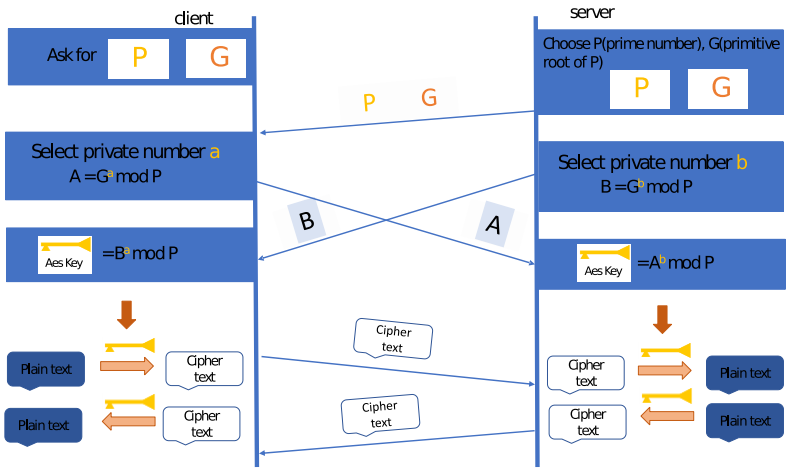
Έπειτα με το module `Random.new().read()` παράγεται ένα τυχαίο μυστικό κλειδί χρήσιμο για την συνάρτηση αποκρυπτογράφησης, δημιουργείται κρυπτογράφημα με την `PKCS1_v1_5.new()` και καλείται η συνάρτηση `cipher.decrypt()` με πρώτο όρισμα την πληροφορία του κρυπτογραφημένου κλειδιού και δεύτερο όρισμα το μυστικό κλειδί εκτελώντας την κύρια διαδικασία.

Το PKCS1 v1.5 ή PKCS1 είναι ένα απλό σχήμα padding που σχεδιάστηκε για χρήση με κλειδιά RSA κατά την κρυπτογράφηση και αποκρυπτογράφηση.

Η διαδικασίας της κρυπτογραφίας μπορεί να συνεχιστεί με τον αλγόριθμο συμμετρικού κλειδιού AES κρυπτογραφώντας και αποκρυπτογραφώντας με το κλειδί aes που τώρα πια διαθέτουν και οι δύο μεριές του συστήματος. Μπορείτε να βρείτε τον κώδικα στο Παράρτημα Α.

### 3.5.3 Ροή αλγορίθμου υβριδικής κρυπτογραφίας με Diffie Hellman και AES

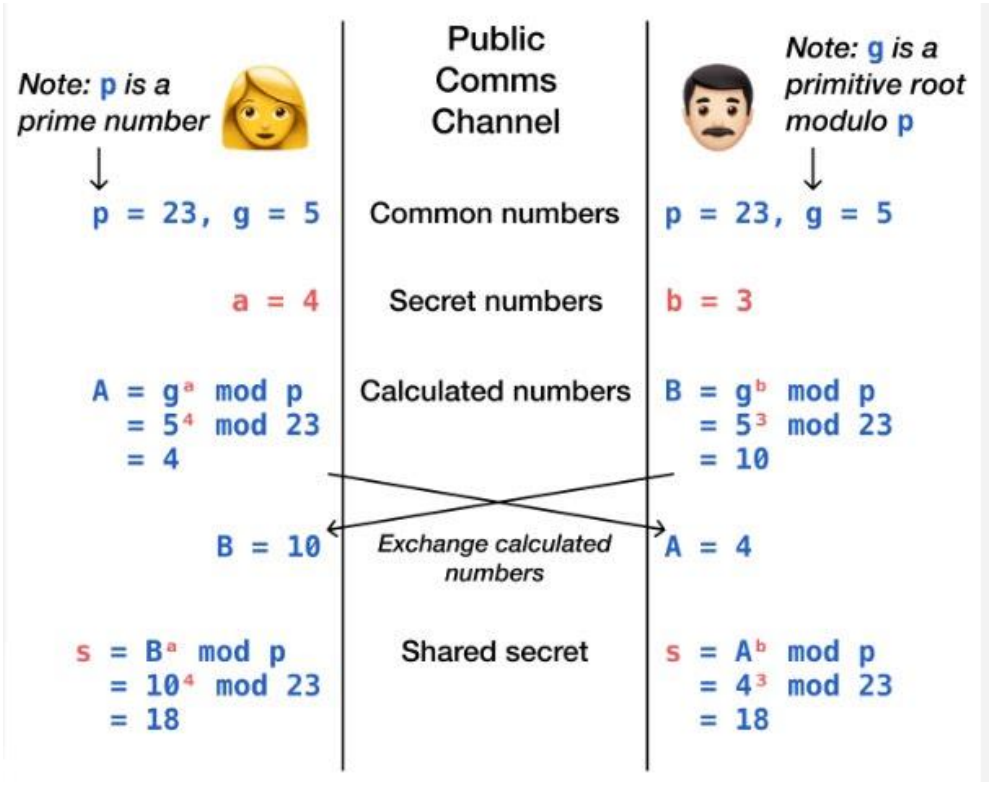
Σε δεύτερη φάση της υλοποιείται ο μηχανισμός ανταλλαγής δημόσιου κλειδιού Diffie Hellman. Αυτό συμβαίνει με σκοπό την αποφυγή της μεταφοράς το κλειδιού AES πάνω από το κανάλι επικοινωνίας ακόμα και στην περίπτωση όπου το μήνυμα μεταφέρεται κρυπτογραφημένο.




*Εικόνα 35: Αλγόριθμος υβριδικής κρυπτογραφίας με DH και AES*

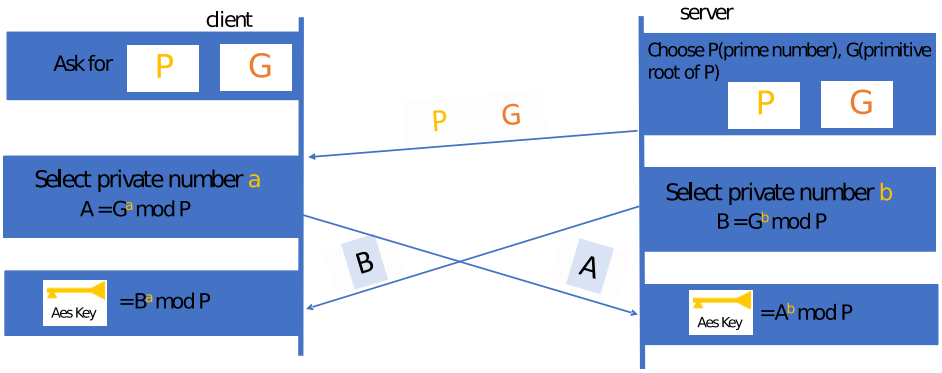
Ο διακομιστής είναι υπεύθυνος για την δημιουργία αυτών των κατάλληλων αριθμών P και G. Ο πελάτης λαμβάνει τα κοινά νούμερα από τον διακομιστή, ζητώντας στέλνοντας το κατάλληλο μήνυμα με σηματοδοσία “P” και “G”. Αφού τα δύο άκρα διαθέτουν τους δύο κοινούς αριθμούς και αφού έχουν επιλέξει τους ιδιωτικούς τυχαίους αριθμούς προχωρούν για τον υπολογισμό των A και B με κατάλληλες μαθηματικές πράξεις. Πελάτης και διακομιστής ανταλλάσσουν τα A και B και με τελικούς υπολογισμούς είναι έτοιμοι να καταλήξουν στο μυστικό κλειδί. Το κλειδί μετατρέπεται σε 256 bits για να μπορέσει να αποτελέσει το κλειδί του αλγορίθμου AES για κρυπτογράφηση και αποκρυπτογράφηση.

### 3.5.4 Ανταλλαγή κλειδιών με Diffie Hellman




*Εικόνα 36: Αλγόριθμος Diffie Hellman*

Όπως φαίνεται και από το παραπάνω σχήμα, τα δύο άκρα συμφωνούν στη χρήση δύο κοινών αριθμών P,G οι οποίοι μπορούν να μεταφερθούν πάνω από το δίκτυο χωρίς κίνδυνο. Ο P είναι ένας πρώτος αριθμός και ο G είναι πρώτη ρίζα του P. Έπειτα με κατάλληλους υπολογισμούς καταλήγουν στο ίδιο μυστικό κλειδί το οποίο θα αποτελέσει και το κλειδί του αλγορίθμου AES.




*Εικόνα 37:Ανταλλαγή κλειδιών με Diffie Hellman*

Ο διακομιστής είναι υπεύθυνος για την δημιουργία αυτών των κατάλληλων αριθμών καλώντας την `primesInRange()` για να επιλέξει έναν τυχαίο πρώτο αριθμό μεταξύ

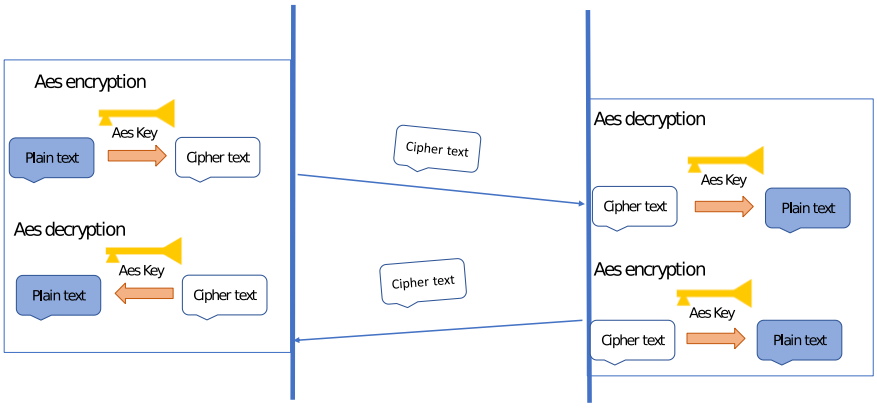
συγκεκριμένου διαστήματος και την `findPrimitiveRoot()` για να βρει την πρώτη ρίζα του αριθμού που έχει επιλεχθεί νωρίτερα. Ο πελάτης λαμβάνει τα κοινά νούμερα από τον διακομιστή, ζητώντας τα με τις συναρτήσεις `askForP()` και `askForB()` στέλνοντας το κατάλληλο μήνυμα με σηματοδοσία "P" και "G". Για να διαβάσει σε σωστά ο πελάτης τον αριθμό που έχει φτάσει δημιουργείται η συνάρτηση `readNumber()`.

Αφού τα δύο άκρα διαθέτουν τους δύο κοινούς αριθμούς και αφού έχουν επιλέξει τους ιδιωτικούς τυχαίους αριθμούς με τη συνάρτηση `random()` προχωρούν για τον υπολογισμό των A και B στη συνάρτηση όπου γίνονται οι κατάλληλες μαθηματικές πράξεις. Τα A και B ανταλλάσσονται καλώντας τις συναρτήσεις `sendA()` και `send()` αντίστοιχα.

Πελάτης και διακομιστής καλώντας τη συνάρτηση `DiffieHellmanFormula()` και κάνοντας τους τελικούς υπολογισμούς είναι έτοιμοι να καταλήξουν στο μυστικό κλειδί. Το κλειδί μετατρέπεται σε 256 bits για να μπορέσει να αποτελέσει το κλειδί του αλγορίθμου AES για κρυπτογράφηση και αποκρυπτογράφηση.

Ο χρήστης μπορεί να επιλέξει την μέθοδο ανταλλαγής κλειδιών που επιθυμεί ύστερα από ερώτηση που του γίνεται πριν ξεκινήσει η βασική διαδικασία.

### 3.5.5 Κρυπτογράφηση και αποκρυπτογράφηση




*Εικόνα 38: Ροή πρωτοκόλλου κρυπτογράφησης και αποκρυπτογράφησης με AES*

### 3.5.5.1 Από την μεριά του πελάτη

Για την κρυπτογράφηση των μηνυμάτων που πρόκειται να σταλθούν καλείται η συνάρτηση `encryptMessage()` στην οποία γίνεται χρήση των συναρτήσεων που προσφέρονται από την βιβλιοθήκη Mbed TLS. Η συνάρτηση δέχεται ως όρισμα το μήνυμα που πρόκειται να κρυπτογραφηθεί, το aes κλειδί, τον πίνακα στον οποίο θα αποθηκευτεί το κρυπτογραφημένο κείμενο και μια μεταβλητή που χρησιμοποιείται για την προσθήκη του ανάλογου συμβόλου που καθορίζει τον τύπο του μηνύματος.

Για την χρήση των παρεχόμενων συναρτήσεων πρέπει να προστεθεί στον κώδικα το header.

```
#include "mbedtls/aes.h"
```

Η διαδικασία ξεκινά με την αρχικοποίηση μεταβλητών που θα χρησιμοποιηθούν στην κρυπτογράφηση. Αμέσως μετά ορίζεται το κλειδί και το διάνυσμα αρχικοποίησης που χρειάζεται στην κρυπτογράφηση aes με mode CBC.

Τελικά καλώντας την `mbedtls_aes_crypt_cbc()` με mode λειτουργίας `MBEDTLS_AES_ENCRYPT` γίνεται η κρυπτογράφηση. Η συνάρτηση καλείται για κάθε block της πληροφορίες μέχρι να κρυπτογραφηθεί όλο το μήνυμα.

Σε περίπτωση που το μήκος του μηνύματος είναι μικρότερο από 16 bytes ή δεν είναι πολλαπλάσιο των 16 bytes τότε γίνεται padding (γέμισμα) με μηδενικά μέχρι να πληροί τις προϋποθέσεις.

Η αποκρυπτογράφηση των δεδομένων που λαμβάνει ο πελάτης υλοποιείται στην συνάρτηση `decryptMessage()` και η διαδικασία είναι πανομοιότυπη με αυτή της κρυπτογράφησης αλλάζοντας το mode της `mbedtls_aes_crypt_cbc()` σε `MBEDTLS_AES_DECRYPT`.

### 3.5.5.2 Από την μεριά του διακομιστή

Για την αποκρυπτογράφηση των δεδομένων από την μεριά του διακομιστή καλείται η συνάρτηση `aes_decryption()` η οποία δέχεται ως όρισμα το κλειδί aes καθώς και την πληροφορία που χρήζει αποκρυπτογράφησης. Αφού οριστεί το IV, το διάνυσμα αρχικοποίησης όπως και στη μεριά του πελάτη απαιτείται η δημιουργία νέου aes αντικειμένου.

Ακολουθεί η αποκρυπτογράφηση της πληροφορίας με την `aes.decrypt()` και η αποθήκευση του αποτελέσματος στο `den_text` .

```
den_text = aes.decrypt(encrypted_data)
```

```
output = den_text.hex()
```

Η κρυπτογράφηση των απαντήσεων του διακομιστή προς τον πελάτη γίνεται στην συνάρτηση `aes_encryption()` με ορίσματα το κλειδί aes και το κείμενο που θα κρυπτογραφηθεί. Αφού γίνει padding των δεδομένων στις περιπτώσεις που χρειάζεται και οριστεί το διάνυσμα αρχικοποίησης δημιουργείται ένα aes αντικείμενο και καλείται η συνάρτηση `aes.encrypt()` αποθηκεύοντας στην μεταβλητή `en_text` την κρυπτογραφημένη πληροφορία. Μπορείτε να βρείτε τον κώδικα στο Παράρτημα Α.

# 3.6 Επιτάχυνση υλικού με τη βιβλιοθήκη mmCAU

Στα πλαίσια αξιολόγησης του αλγορίθμου AES ενεργοποιείται ο συνεπεξεργαστής που διαθέτει η πλατφόρμα για την επιτάχυνση υλικού. Στο SDK πακέτο που ενσωματώθηκε στο MCUXpresso IDE παρέχεται παράδειγμα της βιβλιοθήκης CAU και των συναρτήσεων που αυτή προσφέρει ενεργοποιώντας τον συνεπεξεργαστή επιταχύνοντας τις διαδικασίες κρυπτογράφησης και αποκρυπτογράφησης.

Η κρυπτογράφηση υλοποιείται στη συνάρτηση `mmcau_encrypt_aes_cbc()` η οποία δέχεται έξι ορίσματα, το κλειδί της κρυπτογράφησης, το mode, έναν δείκτη στα δεδομένα που εισάγονται και έναν στον πίνακα που πρόκειται να αποθηκευτούν τα δεδομένα μετά τη διαδικασία, το μήκος των δεδομένων και τέλος το διάνυσμα αρχικοποίησης. Στη συνέχεια αναλόγως του μήκους του κλειδιού υπολογίζονται οι κύκλοι AES.

Έπειτα αρχικοποιείται το κλειδί AES καλώντας την `MMCAU_AES_SetKey()` η οποία με τη σειρά της καλεί την `cau_aes_set_key()` της βιβλιοθήκης CAU που εκτελεί τη διαδικασία. Στη συνέχεια με την πράξη XOR προετοιμάζονται τα δεδομένα για το CBC mode.

Και τέλος καλείται η συνάρτηση `cau_aes_encrypt()` μέσω της `MMCAU_AES_EncryptEcb()`.

Η αποκρυπτογράφηση υλοποιείται στη συνάρτηση `mmcau_decrypt_aes_cbc()` Και η οποία δέχεται παρόμοια ορίσματα με την `mmcau_encrypt_aes_cbc()` και ακολουθεί ο καθορισμός των κύκλων AES και της αρχικοποίησης του κλειδιού όπως συμβαίνει και στην `mmcau_encrypt_aes_cbc()`.

Στη συνέχεια μέσω της `MMCAU_AES_DecryptEcb()` καλείται η `cau_aes_decrypt()` που παρέχεται από το CAU API και εκτελεί την αποκρυπτογράφηση. Τέλος υπολογίζεται η πράξη XOR για την εξαγωγή των δεδομένων με CBC mode.

Αφού καθοριστεί το κείμενο προς κρυπτογράφηση και αποκρυπτογράφηση, τα διανύσματα αρχικοποίησης για τον αλγόριθμο AES γίνεται η προετοιμασία του υλικού.

Και καλείται η συνάρτηση `mmcau_example_task()` στην οποία υπολογίζεται η απόδοση των παραπάνω συναρτήσεων. Ο κώδικας της ενότητας βρίσκεται στο Παράρτημα Β.

# Κεφάλαιο 4. Πειραματική αξιολόγηση

Στο κεφάλαιο αυτό αξιολογείται η απόδοση σε ταχύτητα και κατανάλωση ισχύος του συστήματος εξουσιοδότησης. Οι πιο χρονοβόρες διαδικασίες κατά τη λειτουργία του συστήματος είναι η κρυπτογράφησης και αποκρυπτογράφησης του αλγορίθμου AES και η αξιολόγηση εστιάζεται σε αυτές, αρχικά στην περίπτωση χρήσης αποκλειστικά του λογισμικού και έπειτα με επιτάχυνση υλικού. Ο αλγόριθμος AES υλοποιείται σε λειτουργία CBC με τρία διαφορετικά μήκη κλειδιών των 128, 192 και 256 bits και το κρυπτογράφημα εισόδου είναι 16 bytes. Επιτάχυνση υλικού υλοποιείται με τη βιβλιοθήκη mmCAU και τον Crypto Acceleration συνεπεξεργαστή που διαθέτει η πλατφόρμα FRDM-K64F.

Η μεθοδολογία αξιολόγησης εστιάζοντας στο λογισμικό, αναπτύσσεται στο PlatformIO ενώ ο αλγόριθμος για την χρήση του υλικού γράφεται στο προγραμματιστικό περιβάλλον MCUXpresso IDE και βασίζεται σε πακέτο SDK που παρέχεται από την εταιρεία NXP και ενσωματώθηκε στο προγραμματιστικό περιβάλλον.

## 4.1 Μετρική 1: Χρόνος αρχικοποίησης κλειδιού AES

### 4.1.1 Διαδικασία παραγωγής μετρήσεων

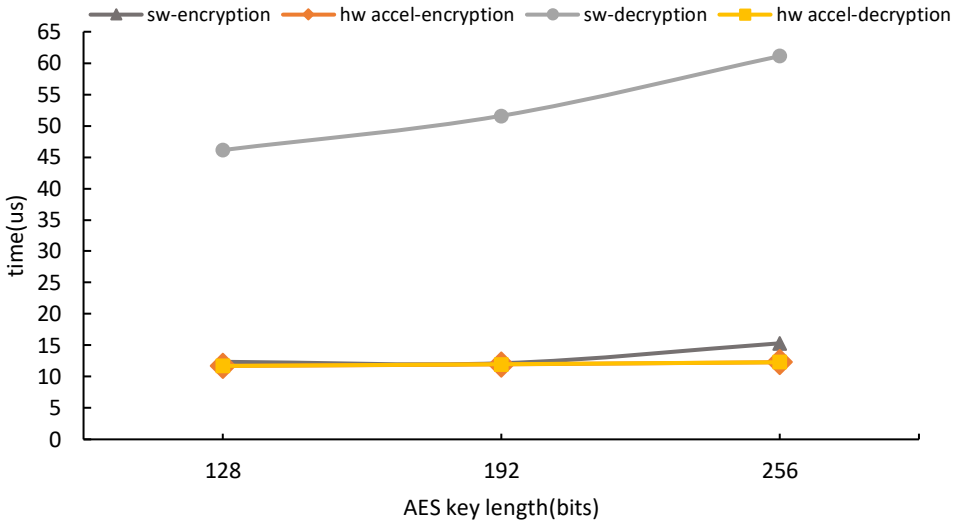
Η πρώτη μετρική που αξιολογείται είναι ο χρόνος που απαιτείται για να καθοριστεί το κλειδί τους αλγορίθμου AES σε microseconds. Για να υπολογισθεί η παραπάνω μετρική ακολουθείται η μεθοδολογία που φαίνεται στον παρακάτω ψευδοκώδικα τόσο κατά την διαδικασία της κρυπτογράφησης όσο και κατά την αποκρυπτογράφηση:

```
While(iterations){
      time.start()
      //Operation under measure
      mbedtls_aes_setkey(&aes, aes_key, length);
      time.stop()
      elapsedTime=time.stop()-time.start();
      totalTime+=elapsedTime
      iterations--
}
```

Τελικό βήμα για τον υπολογισμό του χρόνου αρχικοποίησης του κλειδιού σε μία επανάληψη είναι η διαίρεση totalTime/iterations, με αριθμό επαναλήψεων 1000.

### 4.1.2 Αποτελέσματα

Στο διάγραμμα που ακολουθεί στον άξονα x παρουσιάζονται τα διαφορετικά μήκη κλειδιών κατά την διαδικασία της κρυπτογραφίας ενώ στον κάθετο άξονα ο χρόνος αρχικοποίησης που χρειάστηκε σε κάθε μια περίπτωση. Είναι σαφές πως ο χρόνος αρχικοποίησης κατά την διαδικασία της αποκρυπτογράφησης με τη χρήση του υλικού είναι βελτιωμένος κατά 70% σε σχέση με την ίδια τιμή χρησιμοποιώντας το λογισμικό ενώ μικρή βελτίωση παρουσιάζεται και κατά την διαδικασία την κρυπτογράφησης κυρίως κατά την χρήση του 256 bits κλειδιού AES.




*Εικόνα 39: Συγκρίσεις χρόνου αρχικοποίησης διαφορετικού μεγέθους κλειδιού AES με και χωρίς επιτάχυνση υλικού.*

*Πίνακας 1: Τιμές αρχικοποίησης διαφορετικού μεγέθους κλειδιών AES με και χωρίς τη χρήση επιτάχυνσης υλικού.*

| AES key length(bits) | sw-encryption(us) | hw accel-encryption(us) | sw-decryption(us) | hw accel-decryption(us) |
|---|---|---|---|---|
| 128 | 12,37 | 11,712 | 46,13 | 11,713 |
| 192 | 12,14 | 11,963 | 51,59 | 11,963 |
| 256 | 15,32 | 12,297 | 61,11 | 12,296 |

# 4.2 Μετρική 2: Χρόνος εκτέλεσης διαδικασιών AES

## 4.2.1 Διαδικασία παραγωγής μετρήσεων

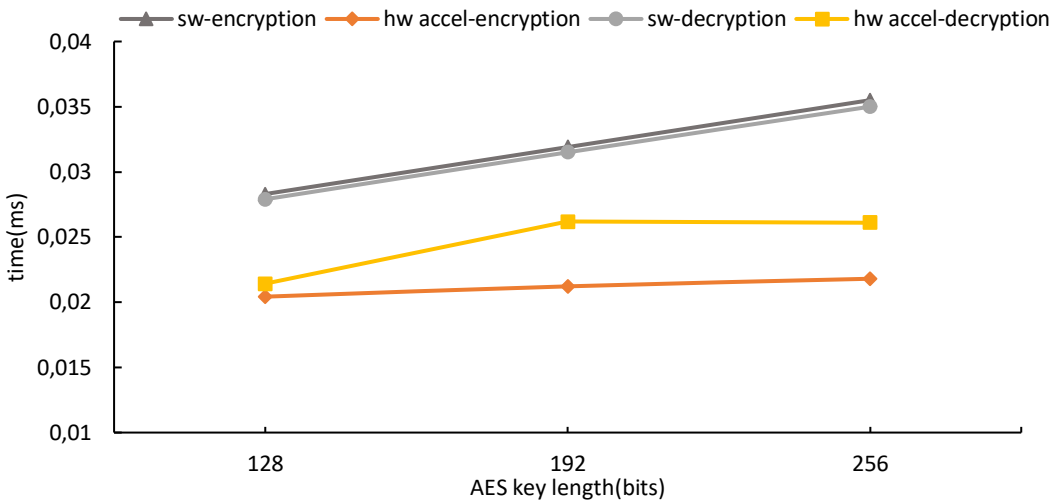
Η μετρική που αξιολογείται είναι ο χρόνος που απαιτείται για την ολοκλήρωση της διαδικασίας κρυπτογράφησης και αποκρυπτογράφησης AES σε milliseconds. Για να προκύψουν τα αποτελέσματα που θα αξιολογήσουν την απόδοση του αλγορίθμου AES με βάση την παραπάνω μετρική εφαρμόστηκε ο παρακάτω ψευδοκώδικας:

```
time.start()
While(iterations){
      //Operation under measure
            mbedtls_aes_crypt_cbc();
      iterations--
}
time.stop()
elapsedTime =time.stop()-time.start();
```

Έπειτα ο χρόνος για μία επανάληψη προκύπτει διαιρώντας τον συνολικό χρόνο με τον αριθμό των επαναλήψεων elaspedTime/iterations, iterations=10000.

### 4.2.2 Αποτελέσματα

Στο γράφημα που ακολουθεί στον άξονα x παρουσιάζονται τα διαφορετικά μήκη κλειδιών κατά την διαδικασία της κρυπτογραφίας ενώ στον κάθετο άξονα ο χρόνος ολοκλήρωσης των υλοποιήσεων AES που χρειάστηκε σε κάθε μια περίπτωση. Σύμφωνα με το που ακολουθεί παρατηρείται μείωση του χρόνου ολοκλήρωσης της διαδικασία της κρυπτογράφησης και αποκρυπτογράφησης κατά την επιτάχυνση υλικού σε milliseconds. Ο χρόνος αυξάνεται σχεδόν γραμμικά στην περίπτωση της χρήσης λογισμικού όσο αυξάνεται και το μήκους του κλειδιού AES. Με τη χρήση του υλικού ο χρόνος εκτέλεσης της κάθε διαδικασίας μειώνεται αισθητά. Στην περίπτωση της κρυπτογράφησης παρατηρείται βελτίωση έως και 40% ενώ ο χρόνος εκτέλεσης της αποκρυπτογράφησης μειώθηκε έως και 25% στην περίπτωση χρήσης 256 bits κλειδιού AES.




*Εικόνα 40: Συγκρίσεις χρόνου ολοκλήρωσης των υλοποιήσεων του αλγορίθμου AES με χρήση λογισμικού αλλά και με επιτάχυνσης υλικού.*

*Πίνακας 2: Τιμές χρόνων ολοκλήρωσης των υλοποιήσεων του αλγορίθμου AES με χρήση λογισμικού αλλά και με επιτάχυνσης υλικού.*

| AES key length(bits) | sw-encryption(ms) | hw accel-encryption(ms) | sw-decryption(ms) | hw accel-decryption(ms) |
|---|---|---|---|---|
| 128 | 0,0283 | 0,020426 | 0,0279 | 0,021426 |
| 192 | 0,0319 | 0,021218 | 0,0315 | 0,026194 |
| 256 | 0,0355 | 0,021801 | 0,035 | 0,026111 |

# 4.3 Μετρική 3: Κύκλοι ρολογιού επεξεργαστή ανά byte δεδομένων

## 4.3.1 Διαδικασία παραγωγής μετρήσεων

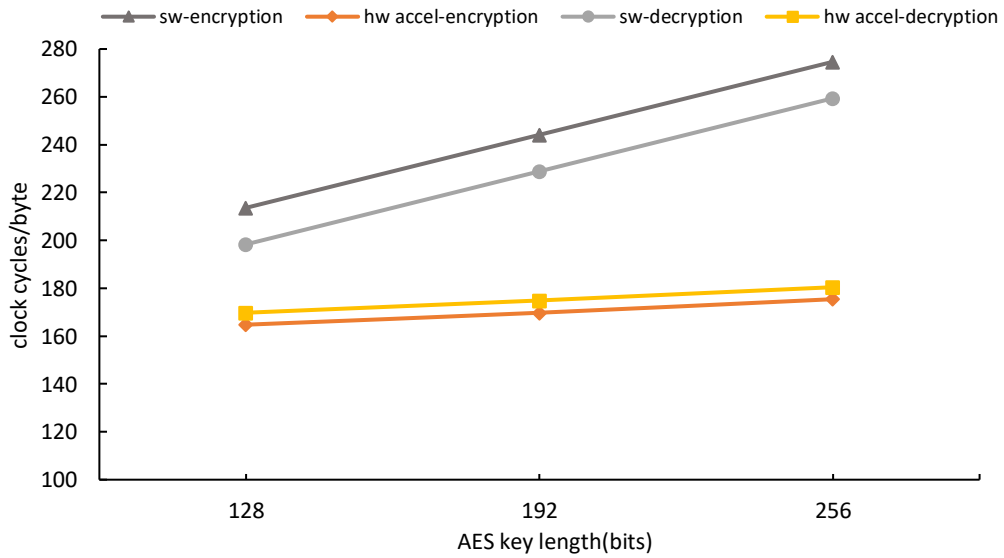
Με την μετρική κύκλοι ρολογιού επεξεργαστή ανά byte δεδομένων υπολογίζεται ο αριθμός των κύκλων ρολογιού που εκτελεί ο μικροεπεξεργαστής ανά byte δεδομένων για κάθε μία από τις υλοποιήσεις του AES συγκρίνοντας τις δύο μεθοδολογίες χρησιμοποιώντας τα διαφορετικά μήκη κλειδιών. Χρειάστηκε να υπολογισθούν οι κύκλοι ανά δευτερόλεπτο καθώς και τα byte δεδομένων που μπορούν να

επεξεργαστούν σε ένα δευτερόλεπτο, δηλαδή η ρυθμαπόδοση. Εφαρμόστηκε η εξής σχέση αντικαθιστώντας τις τιμές που έχουν ήδη υπολογισθεί:

```
Cycles per second= 120MHz
Cycles per byte= cycles per second/byte per second=
(120 * 10⁶) / (throughput * 1024)
```

### 4.3.2 Αποτελέσματα

Στον άξονα x του παρακάτω γραφήματος παρουσιάζονται τα διαφορετικά μήκη κλειδιών του AES σε bits. Στον κάθετο άξονα βρίσκονται οι κύκλοι ρολογιού του επεξεργαστή. Οι κύκλοι που χρειάζονται κάθε φορά είτε με τη χρήση λογισμικού είτε με την επιτάχυνση υλικού αυξάνονται γραμμικά όσο αυξάνεται και το μήκος του κλειδιού AES. Με τη χρήση υλικού μειώνεται αρκετά ο αριθμός των κύκλων ρολογιού του επεξεργαστή ανά byte δεδομένων σε κάθε υλοποίηση του AES γεγονός που βοηθά στην καλύτερη απόδοσή του συστήματος. Στην περίπτωση χρήσης κλειδιού μεγέθους 128 bits η βελτίωση είναι περίπου 20% ενώ όσο το μήκος του κλειδιού αυξάνεται η κύκλοι ανά byte πέφτουν αισθητά φτάνοντας έως και μείωση 40%.




*Εικόνα 41: Συγκρίσεις κύκλων ρολογιού του επεξεργαστή ανά byte δεδομένων κατά τις υλοποιήσεις του AES με και χωρίς τη χρήση επιτάχυνσης υλικού.*

*Πίνακας 3: Τιμές κύκλων ρολογιού του επεξεργαστή ανά byte δεδομένων κατά τις υλοποιήσεις του AES με και χωρίς τη χρήση επιτάχυνσης υλικού*

| AES key length(bits) | sw-encryption | hw accel-encryption | sw-decryption | hw accel-decryption |
|---|---|---|---|---|
| 128 | 213,623 | 164,774 | 198,364 | 169,770 |
| 192 | 244,141 | 169,776 | 228,882 | 174,774 |
| 256 | 274,658 | 175,44 | 259,399 | 180,447 |

# 4.4 Μετρική 4: Ρυθμαπόδοση

## 4.4.1 Διαδικασία παραγωγής μετρήσεων

Η ρυθμαπόδοση(throughput) αναφέρεται στον όγκο δεδομένων που μπορούν να επεξεργαστούν στην μονάδα του χρόνου. Συγκεκριμένα υπολογίζεται σε megabytes ανά δευτερόλεπτο. Η μεθοδολογία που εφαρμόζεται για την εξαγωγή των μετρήσεων είναι η εξής:

```
time.start()
While(iterations){
//Operation under measure
mbedtls_aes_crypt_cbc();
iterations--
}
time.stop()
elapsedTime_for_throughput=time.stop()-time.start();
```

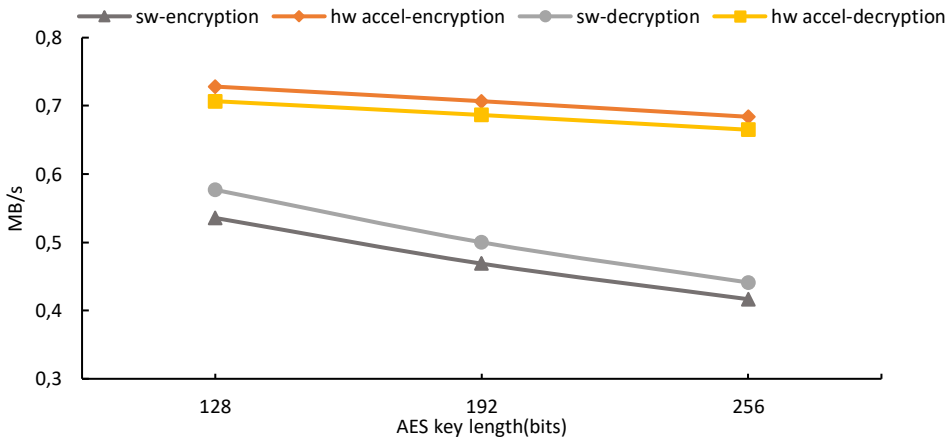
Εν συνεχεία, γίνεται αντικατάσταση των τιμών που υπολογίσθηκαν προηγουμένως στην παρακάτω σχέση ώστε να προκύψει η τιμή της ρυθμαπόδοσης.

```
throughput = (1000* bytes of plaintext)/1024/1024)/elapsedTime
```

Η παραπάνω μεθοδολογία ακολουθείται και στις δύο κρυπτογραφικές διαδικασίες.

### 4.4.2 Αποτελέσματα

Στο διάγραμμα που ακολουθεί στον άξονα x παρουσιάζονται τα διαφορετικά μήκη κλειδιών κατά την διαδικασία της κρυπτογραφίας ενώ στον κάθετο άξονα τα megabytes ανά δευτερόλεπτο. Από τα αποτελέσματα που προκύπτουν συμπεραίνεται ότι η ρυθμαπόδοση σημειώνει αρκετά μεγάλη βελτίωση, της τάξεως του 30%, με τη χρήση του Crypto Acceleration συνεπεξεργαστή τόσο κατά την διαδικασία της κρυπτογράφησης όσο και κατά την διαδικασία της αποκρυπτογράφησης. Στο διάγραμμα που ακολουθεί γίνεται αντιληπτή η παραπάνω παρατήρηση με καλύτερη ρυθμαπόδοση να προκύπτει κατά την διαδικασία της κρυπτογράφησης αγγίζοντας τα 0.73MB/second.




*Εικόνα 42: Σύγκριση τιμών ρυθμαπόδοσης του αλγορίθμου AES τόσο κατά τη χρήση επιτάχυνσης υλικού όσο και χωρίς.*

*Πίνακας 4:Τιμές ρυθμαπόδοσης του αλγορίθμου AES τόσο κατά τη χρήση επιτάχυνσης υλικού όσο και χωρίς.*

| AES key length | sw-encryption(MB/s) | hw accel-encryption(MB/s) | sw-decryption(MB/s) | hw accel-decryption(MB/s) |
|---|---|---|---|---|
| 128 | 0,53571 | 0,72827 | 0,57692 | 0,706837 |
| 192 | 0,46875 | 0,706815 | 0,5 | 0,686602 |
| 256 | 0,41667 | 0,683997 | 0,44118 | 0,665016 |

# 4.5 Μετρική 5: Κατανάλωση ισχύος

## 4.5.1 Διαδικασία παραγωγής μετρήσεων

Για την αξιολόγηση του αλγορίθμου από την σκοπιά της κατανάλωσης που σημειώνει χρειάστηκε η χρήση USB που μετρά τόσο το ρεύμα όσο και την τάση που παρατηρείται στη συσκευή. Η ισχύς υπολογίζεται με βάση τη σχέση: $P=I*V$.


*Εικόνα 43: USB για μέτρηση ρεύματος και τάσης*

Η διαδικασία που βρίσκεται υπό μέτρηση τοποθετείται σε μια ατέρμονη επανάληψη ενώ παράλληλα καταγράφονται οι τιμές ρεύματος και τάσης που παρατηρούνται στο USB. Το πείραμα γίνεται 10 φορές και υπολογίζεται ο μέσος όρος των τιμών που προκύπτουν. Ο ψευδοκώδικας που ακολουθεί εφαρμόστηκε για να επιτευχθεί η παραπάνω μέτρηση:

```
While(true){
        //Operation under measure
        mbedtls_aes_crypt();
}
```

Η παραπάνω μεθοδολογία ακολουθήθηκε για την εξαγωγή αποτελεσμάτων τόσο κατά την χρήση λογισμικού όσο και κατά την χρήση του υλικού για κρυπτογράφηση και αποκρυπτογράφηση με μήκη κλειδιών 128, 192, 256 bits.

Η τιμή της μέσης κατανάλωσης ανά περίοδο ενεργοποίησης προκύπτει από την εξής σχέση:

$P\_average\_per\_period=P*t+(T-t)*P_{idle})/T$

Όπου $P$: η ισχύς της εκάστοτε διαδικασίας

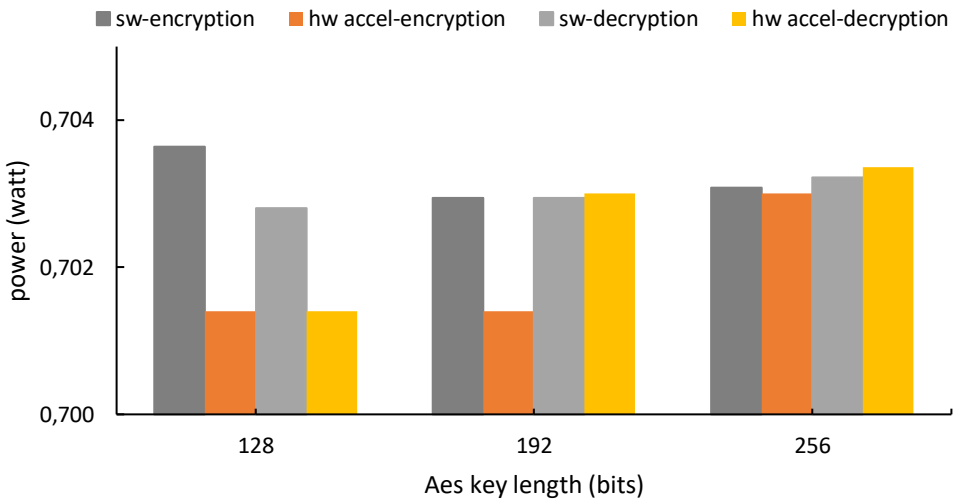
$t$: ο χρόνος εκτέλεσης της διαδικασίας

$T$: η περίοδος ενεργοποίησης της διαδικασίας

$P_{idle}$: η ισχύς του μικροελεγκτή σε αδράνεια. Η τιμή αυτή είναι ίδια και στις δύο περιπτώσεις υπό αξιολόγηση και έχει υπολογιστεί σε 0,4518 watt

### 4.5.2 Αποτελέσματα

Στο άξονα x του παρακάτω γραφήματος τοποθετούνται τα διαφορετικά μήκη κλειδιών σε bits ενώ στον άξονα y οι τιμές της κατανάλωσης ισχύος σε watt. Η κατανάλωση ισχύος κυμαίνεται από 0,701-0,703 watt. Από το γράφημα συμπεραίνεται ότι στην περίπτωση χρήσης 128 bits κλειδιού η κατανάλωση ισχύος βελτιώνεται κατά παραμένει σχεδόν αμετάβλητη είτε κατά την αποκλειστική χρήση λογισμικού είτε κατά την επιτάχυνση υλικού.



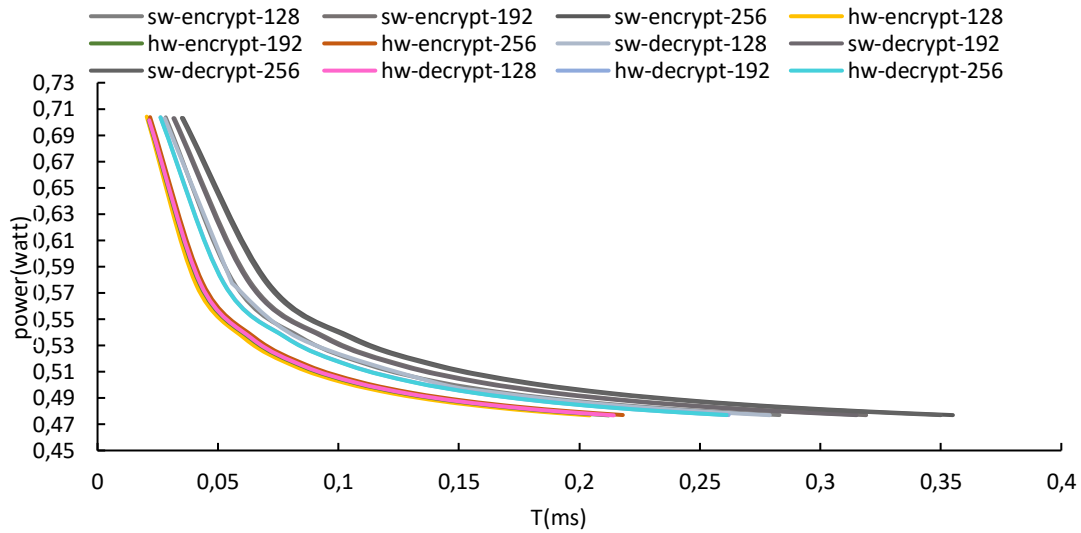

*Εικόνα 44: Σύγκριση κατανάλωσης ισχύος κατά τη χρήση αποκλειστικά λογισμικού αλλά και κατά τη χρήση επιτάχυνσης υλικού των υλοποιήσεων του AES*

*Πίνακας 5: Τιμές κατανάλωσης ισχύος κατά τη χρήση αποκλειστικά λογισμικού αλλά και κατά τη χρήση επιτάχυνσης υλικού των υλοποιήσεων του AES*

| Aes key length (bits) | sw-encryption(watt) | hw accel-encryption(watt) | sw-decryption(watt) | hw accel-decryption(watt) |
|---|---|---|---|---|
| 128 | 0,704 | 0,701 | 0,703 | 0,701 |
| 192 | 0,703 | 0,701 | 0,703 | 0,703 |
| 256 | 0,703 | 0,703 | 0,703 | 0,703 |

Έπειτα οι καμπύλες του γραφήματος που ακολουθεί αναπαριστούν την ενέργεια της κάθε υλοποίησης AES ανά περίοδο ενεργοποίησης τόσο κατά την επιτάχυνση υλικού

όσο και χωρίς. Στον άξονα x σημειώνεται η περίοδος ενεργοποίησης της εκάστοτε διαδικασίας ενώ στον κάθετο άξονα η ισχύς που καταναλώνεται σε κάθε περίπτωση. Γίνεται αντιληπτό πως όσο μειώνεται η συχνότητα ενεργοποίησης της διαδικασίας, η τιμή της κατανάλωσης ισχύος τείνει στην τιμή της κατανάλωσης ισχύος όταν το σύστημα βρίσκεται σε κατάσταση ηρεμίας. Στον άξονα x οι καμπύλες ξεκινούν από την μέγιστη δυνατή τιμή ενεργοποίησης της διαδικασίας.




*Εικόνα 45: Κατανάλωση ισχύος του συστήματος ανά περίοδο ενεργοποίησης κατά τις υλοποιήσεις του AES με και χωρίς τη χρήση επιτάχυνσης υλικού.*

# 4.6 Μετρική 6: Κατανάλωση Ενέργεια

## 4.6.1 Διαδικασία παραγωγής μετρήσεων

Έπειτα οι μετρήσεις της κατανάλωσης ισχύος που συλλέχθηκαν χρησιμοποιούνται για τον υπολογισμό της κατανάλωσης ενέργειας του συστήματος και αντικαθιστούνται στην παρακάτω σχέση:

$E=P*t$

Όπου $P$: η κατανάλωση ισχύος της εκάστοτε διαδικασίας

$t$: ο χρόνος εκτέλεσης της διαδικασίας

Για να υπολογιστεί η ενέργεια που καταναλώνεται σε κάθε μία από τις διαδικασίες ανάλογα με την περίοδο ενεργοποίησης η σχέση τροποποιείται ως εξής:

$E=P\_average\_per\_period*t=[(P*t+(T-t)*P_{idle})/T]*t$

Όπου *P*: η ισχύς της εκάστοτε διαδικασίας

*t*: ο χρόνος εκτέλεσης της διαδικασίας

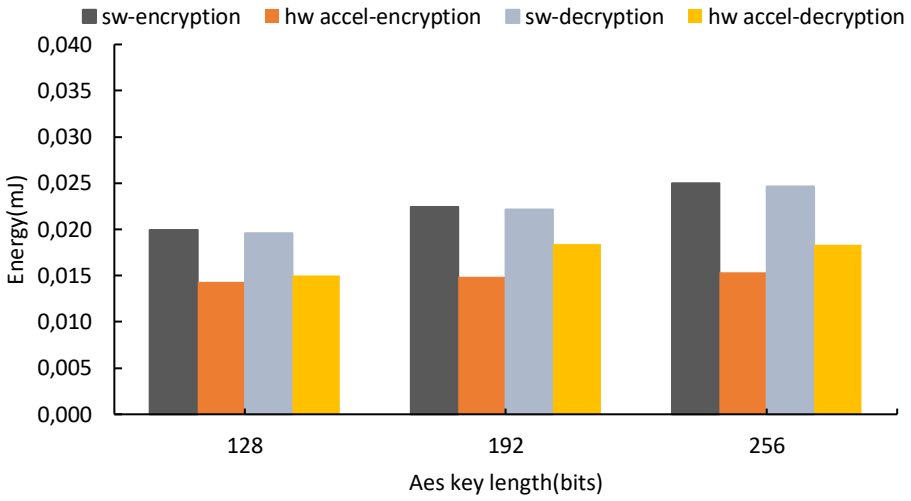
*T*: η περίοδος ενεργοποίησης της διαδικασίας

$P_{idle}$: η ισχύς του μικροελεγκτή σε αδράνεια. Η τιμή αυτή είναι ίδια και στις δύο περιπτώσεις υπό αξιολόγηση και έχει υπολογιστεί σε 0,4518 watt

Ο όρος $(P*t+(T-t)*P_{idle})$ είναι η μέση κατανάλωση ισχύος ανά περίοδο ενεργοποίησης

### 4.6.2 Αποτελέσματα

Στο άξονα x του παρακάτω γραφήματος τοποθετούνται τα διαφορετικά μήκη κλειδιών σε bits ενώ στον άξονα y οι τιμές της ενέργειας σε mJoule. Αναπαρίσταται η ενέργεια που καταναλώνεται στην περίπτωση της κρυπτογράφησης και αποκρυπτογράφησης συγκρίνοντας τις διαφορετικές μεθοδολογίες που βρίσκονται υπό αξιολόγηση. Στην περίπτωση χρήση λογισμικού οι τιμές της κατανάλωσης ενέργειας του συστήματος είναι εμφανώς υψηλότερες και αυξάνονται όσο αυξάνεται και το μέγεθος του κλειδιού AES. Στην περίπτωση της υλοποίησης του AES με επιτάχυνση υλικού η ενέργεια που καταναλώνεται είναι εμφανώς μικρότερη και αυτό αποδεικνύει ότι η χρήση της επιτάχυνσης υλικού βελτιώνει την απόδοση του συστήματος. Στην περίπτωση της χρήσης 128 bits κλειδιού παρατηρείται βελτίωση περίπου 30%. Στην περίπτωση χρήσης 192 bits κλειδιού παρατηρείται βελτίωση 30% στην περίπτωση της κρυπτογράφησης και 20% στην περίπτωση της αποκρυπτογράφησης. Με τη χρήση 256 bits κλειδιού βελτιώνεται κατά 40% η ενέργεια κατά την κρυπτογράφηση και περίπου 30% στην περίπτωση της αποκρυπτογράφησης.

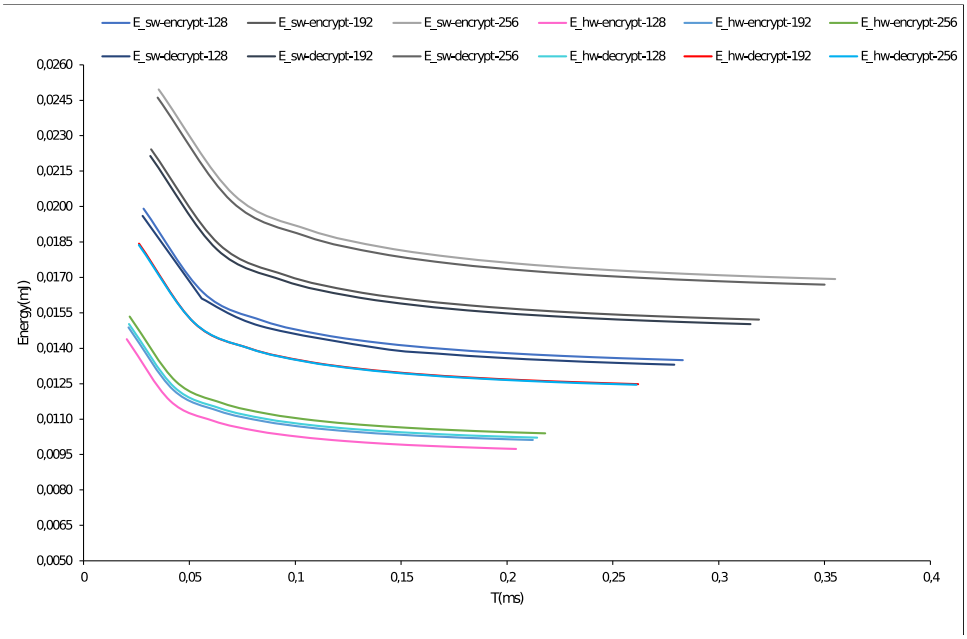



*Εικόνα 46: Συγκρίνεται η κατανάλωση ενέργεια του συστήματος κατά τις υλοποιήσεις του AES συναρτήσει των διαφορετικών μεγεθών κλειδιά κατά τη χρήση επιτάχυνσης υλικού αλλά και χωρίς.*

*Πίνακας 6: Τιμές κατανάλωσης ενέργειας του συστήματος κατά τις υλοποιήσεις του AES σε συνάρτηση με τα διαφορετικά μεγέθη κλειδιών κατά τη χρήση επιτάχυνσης υλικού αλλά και χωρίς.*

| AES key length(bits) | sw-encryption(mJ) | hw accel-encryption(mJ) | sw-decryption(mJ) | hw accel-decryption(mJ) |
|---|---|---|---|---|
| 128 | 0,020 | 0,014 | 0,020 | 0,015 |
| 192 | 0,022 | 0,015 | 0,022 | 0,018 |
| 256 | 0,025 | 0,015 | 0,025 | 0,018 |

Έπειτα οι καμπύλες του γραφήματος που ακολουθεί αναπαριστούν την κατανάλωση ενέργειας της κάθε διεργασίας AES ανά περίοδο ενεργοποίησης τόσο κατά την επιτάχυνση υλικού όσο και χωρίς. Στον άξονα x σημειώνεται η περίοδος ενεργοποίησης της εκάστοτε διαδικασίας ενώ στον κάθετο άξονα η ενέργεια που καταναλώνεται σε κάθε περίπτωση. Όσο μειώνεται η συχνότητα ενεργοποίησης της διαδικασίας, μειώνεται η μέση κατανάλωση ισχύος . Φυσικό επακόλουθο είναι να μειωθεί και η τιμή της ενέργειας που καταναλώνεται η οποία τείνει στην τιμή της ενέργειας του συστήματος όταν αυτό βρίσκεται σε κατάσταση ηρεμίας.




*Εικόνα 47:Κατανάλωση ενέργειας ανά περίοδο ενεργοποίησης κατά τις υλοποιήσεις του AES με και χωρίς τη χρήση επιτάχυνσης υλικού.*

# Κεφάλαιο 5. Επίλογος

Η τεχνολογική εξέλιξη έχει οδηγήσει στην ανάπτυξη του Internet of Things ή Δίκτυο των Πραγμάτων. Η τεχνολογία αυτή έχει βρει εφαρμογή σε όλους τους τομείς την ανθρώπινης ζωής, αναθεωρώντας και την ασφάλεια χώρων με ειδικά σχεδιασμένα συστήματα εξουσιοδότησης και συγκεκριμένα συστήματα εξουσιοδοτημένης πρόσβασης σε χώρους. Αυτά τα συστήματα εγγυόνται ασφάλεια ελέγχοντας την είσοδο ενός ανθρώπου στο χώρο επιδεικνύοντας κάρτες πρόσβασης.

Πιο συγκεκριμένα, σκοπός της εργασίας είναι η υλοποίηση ενός ασφαλούς συστήματος εξουσιοδοτημένης πρόσβασης σε χώρους με κάρτες RFID, η κωδικοποιημένη επικοινωνία των IoT συσκευών με χρήση κρυπτογραφικών συναρτήσεων και ασφαλών πρωτοκόλλων ανταλλαγής κρυπτογραφικών κλειδιών καθώς και η αξιολόγηση της απόδοσης των κρυπτογραφικών διαδικασιών του συστήματος με ή χωρίς επιτάχυνση υλικού. Το σύστημα αποτελείται από έναν διακομιστή, τον μικροελεγκτή FRDM-K64F καθώς και το RFID σύστημα MIFARE RC522. Το λογισμικό του μικροελεγκτή βασίζεται στο λειτουργικό σύστημα πραγματικού χρόνου Mbed ενώ ο διακομιστής προγραμματίζεται σε Python. Το λογισμικό οδήγησης που χρησιμοποιείται για την επικοινωνία του μικροελεγκτή με το σύστημα RFID είναι το MFRC522.

Βασική ανάγκη του συστήματος εξουσιοδότησης είναι η ασφαλή ανταλλαγή πληροφοριών μεταξύ των IoT συσκευών προς αποφυγή κακόβουλων επιθέσεων. Έτσι, αναπτύσσεται υβριδικός κρυπτογραφικός αλγόριθμος συνδυάζοντας τον αλγόριθμο Rivest-Shamir-Adleman (RSA) για την ανταλλαγή των κρυπτογραφικών κλειδιών και τον αλγόριθμο Advanced Encryption Standard (AES) για την κρυπτογραφία των μηνυμάτων. Σε δεύτερη φάση για την ανταλλαγή κλειδιών υλοποιείται ο αλγόριθμος Diffie Hellman (DH). Οι παραπάνω υλοποιήσεις γίνονται κάνοντας χρήση της βιβλιοθήκης mbedtls και κρυπτογραφικών βιβλιοθηκών της Python.

Οι πιο συχνές, και άρα πιο χρονοβόρες διαδικασίες του συστήματος, είναι οι κλήσεις του AES. Έτσι, αξιολογούνται οι κρυπτογραφικές διεργασίες του AES συναρτήσει των

διαφορετικών μεγέθους κλειδιών χρησιμοποιώντας αποκλειστικά λογισμικό αλλά και με επιτάχυνση υλικού ενεργοποιώντας τον συνεπεξεργαστή της πλατφόρμας με χρήση της βιβλιοθήκης mmCAU. Εξετάζεται η κατανάλωση ισχύος και ενέργειας, η ρυθμαπόδοση του συστήματος όπως και ο χρόνος αρχικοποίησης των διαφορετικών μεγέθους κλειδιών AES, ο χρόνος εκτέλεσης της κρυπτογράφησης και αποκρυπτογράφησης αλλά και οι κύκλοι ρολογιού του επεξεργαστή ανά byte δεδομένων σε συνάρτηση του μεγέθους των κλειδιών του AES. Η βελτίωση που σημειώνεται κατά την αρχικοποίηση του κλειδιού AES είναι περίπου 70% με τη χρήση επιτάχυνσης υλικού. Ο χρόνος ολοκλήρωσης των υλοποιήσεων του AES βελτιώνεται από 0,0319ms με αποκλειστική χρήση λογισμικού σε 0,0212ms με χρήση του συνεπεξεργαστή. Η ρυθμαπόδοση σημειώνει 30% βελτίωση αφού κατά τη χρήση λογισμικού υπολογίζεται 0.47MB/s ενώ η αντίστοιχη τιμή με επιτάχυνση υλικού είναι 0.7 MB/s. Οι τιμές της κατανάλωσης ισχύος μπορεί να είναι από 0,701 έως 0,704 watt ενώ της ενέργειας που κυμαίνονται 0,020 έως 0,025 mJoule με τη χρήση αποκλειστικά λογισμικού και 0,015 έως 0,018 mJoule με τη χρήση επιτάχυνσης υλικού.

# Κεφάλαιο 6. Αναφορές


[1] https://en.wikipedia.org/wiki/Internet_of_things

[2] https://www.leverege.com/iot-ebook/how-iot-systems-work

[3] https://os.mbed.com/media/uploads/GregC/frdm-k64f_ug_rev0.1.pdf

[4] https://el.wikipedia.org/wiki/RFID

[5] https://en.wikipedia.org/wiki/Radio-frequency_identification

[6] https://www.abr.com/passive-rfid-tags-vs-active-rfid-tags/

[7]https://medium.com/autonomous-robotics/an-introduction-to-rfid-dc6228767691

[8] https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

[9] https://os.mbed.com/mbed-os/

[10] https://os.mbed.com/docs/mbed-os/v6.6/introduction/index.html

[11] https://platformio.org/

[12] https://mcuxpresso.nxp.com/en/welcome

[13]https://el.wikipedia.org/wiki/%CE%9A%CF%81%CF%85%CF%80%CF%84%CE%BF%CE%B3%CF%81%CE%AC%CF%86%CE%B7%CF%83%CE%B7

[14] https://cgi.di.uoa.gr/~klimn/cryptography/Lab/Lab-6.pdf

[15] https://cybernews.com/resources/what-is-aes-encryption/

[16] https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/

[17] https://mbed-tls.readthedocs.io/en/latest/kb/how-to/generate-an-aes-key/

[18]https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[19] https://en.wikipedia.org/wiki/Hybrid_cryptosystem

[20]https://mbed-tls.readthedocs.io/en/latest/kb/how-to/mbedtls-tutorial/

[21] https://en.wikipedia.org/wiki/Transport_Layer_Security

[22] https://mbed-tls.readthedocs.io/en/latest/

[23] https://www.nxp.com/docs/en/user-guide/CAUAPIUG.pdf

[24]https://helpdeskgeek.com/reviews/what-is-hardware-acceleration-and-how-is-it-useful/

[25] https://www.geeksforgeeks.org/client-server-model/

Boano, C.A., Römer, K., Bloem, R. et al. Dependability for the Internet of Things—from dependable networking in harsh environments to a holistic view on dependability. Elektrotech. Inftech. 133, 304–309 (2016).

F. Shao, Z. Chang and Y. Zhang, "AES Encryption Algorithm Based on the High Performance Computing of GPU," 2010 Second International Conference on Communication Software and Networks, 2010, pp. 588-590, doi: 10.1109/ICCSN.2010.124.

M. J. Wiener, "Cryptanalysis of short RSA secret exponents," in IEEE Transactions on Information Theory, vol. 36, no. 3, pp. 553-558, May 1990, doi: 10.1109/18.54902.

A. K. Mandal, C. Parakash and A. Tiwari, "Performance evaluation of cryptographic algorithms: DES and AES," 2012 IEEE Students' Conference on Electrical, Electronics and Computer Science, 2012, pp. 1-5, doi: 10.1109/SCEECS.2012.6184991.

S. Narang, T. Nalwa, T. Choudhury and N. Kashyap, "An efficient method for security measurement in internet of things," 2018 International Conference on Communication, Computing and Internet of Things (IC3IoT), 2018, pp. 319-323, doi: 10.1109/IC3IoT.2018.8668159.

D. Punia and B. Singh, "Speed Optimization of the AES Algorithm Using Pipeline Hardware Architecture," 2019 International Conference on Communication and Electronics Systems (ICCES), 2019, pp. 2070-2074, doi: 10.1109/ICCES45898.2019.9002086.

## ΠΑΡΑΡΤΗΜΑ Α

### client.cpp

```
// Test of cheap 13.56 Mhz RFID-RC522 module from eBay
// This code is based on Martin Olejar's and Thomas Kirchner's
MFRC522 libraries. Minimal changes
// Adapted for FRDM-K64F from Freescale, in 07/21/2014 by Clovis
```

Fritzen.

```
#include "mbed.h"
#include "MFRC522.h"
#include "EthernetInterface.h"
#include "mbed_config.h"
#include <string.h>
#include <stdio.h>
#include "mbedtls/error.h"
#include "mbedtls/pk.h" //gg
#include "mbedtls/entropy.h"
#include "mbedtls/ctr_drbg.h"
#include "mbedtls/version.h"
#include "mbedtls/aes.h"
#include <time.h>
#include <sys/time.h>

#define FAIL_TO_RECEIVE_DIFF_HELLMAN_PARAM -1000
#define DIFFIE_HELLMAN 1

/* MESSAGE SYMBOLS MEANING
! --> CLIENT ASK FOR PUBLIC KEY
# --> DEVIDE ID
@ --> CARD ID
^ --> ENCREPTYD AES KEY
P --> P DIFFIE HELLMAN PARAMETER
G --> G DIFFIE HELLMAN PARAMETER
B --> B DIFFIE HELLMAN PARAMETER
* --> CHOOSE KEY EXCHANGE METHOD
*/

// FRDM-K64F (Freescale) Pin for MFRC522 reset
#define MF_RESET PTD0

// Serial connection to PC for output*/
static BufferedSerial pc(PTC17, PTC16); // serial comm on the FRDM
board

//  MFRC522       RfChip     (SPI_MOSI,  SPI_MISO,  SPI_SCK,  SPI_CS,
MF_RESET);
MFRC522 RfChip(PTD2, PTD3, PTD1, PTE25, PTD0);

EthernetInterface net;
DigitalOut LedGreen(LED2);
DigitalOut LedBlue(LED3);
DigitalOut LedRed(LED1);

struct message
{ // Structure declaration
  char type;
  int length;
  char *payload;
};

void turnOnBlueLight()
{
  LedGreen = 1; // off
  LedBlue = 0;  // on
  LedRed = 1;   // off
}
void turnOnRedLight()
```

```
{
  LedGreen = 1; // off
  LedBlue = 1;  // off
  LedRed = 0;   // on
}

void turnOnGreenLight()
{
  LedGreen = 0; // on
  LedBlue = 1;  // off
  LedRed = 1;   // off
}

void turnOffAllLights()
{
  LedGreen = 1; // off
  LedBlue = 1;  // off
  LedRed = 1;   // off
}

void findUniqueDeviceId(char dev_id[])
{

  // 4 registers with 32 bit- 4 bytes each and all together gives 128
bit unique id of the device- see manual page 305
  int IdentificationRegistersAddresses[] = {0x40048054, 0x40048058,
0x4004805C, 0x40048060};
  int i;
  for (i = 0; i < 4; i++)
  {
    char *pointer = (char *)IdentificationRegistersAddresses[i];

    dev_id[i * 4] = pointer[0];
    dev_id[(i * 4) + 1] = pointer[1];
    dev_id[(i * 4) + 2] = pointer[2];
    dev_id[(i * 4) + 3] = pointer[3];
  }
  /*
    for (int i = 0; i < 16; ++i) {
        printf("%x", temp[i]);
    }
  printf("\r\n");*/
}

void print_id(char *array, int n)
{
  printf("%.*s ", 1, array);
  for (int i = 1; i < n; i++)
  {
    printf("%X ", array[i]);
  }
  // printf("\r\n");
}

void print_array(unsigned char *array, int n)
{

  for (int i = 0; i < n; i++)
  {
    printf("%X ", array[i]);
  }
```

```
  printf("\r\n");
}

void print_byte_array(char *array, int n)
{
  for (int i = 0; i < n; i++)
  {
    printf("%X ", array[i]);
  }
  printf("\r\n");
}
bool decryptMessage(char chipherText[], unsigned char aes_key[], int
isCard)
{

  unsigned char allow[] = {0x31};
  mbedtls_aes_context aes;
  mbedtls_aes_init(&aes);
  mbedtls_aes_setkey_dec(&aes, aes_key, 256); // 32 bytes key

  size_t INPUT_LENGTH = 16;
  unsigned char decrypt_output[16];

  unsigned char iv[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0e};
  mbedtls_aes_crypt_cbc(&aes, MBEDTLS_AES_DECRYPT, INPUT_LENGTH, iv,
(const unsigned char *)chipherText, decrypt_output);

  mbedtls_aes_free(&aes);
  printf("DECRYPT OUTPUT:");
  print_array(decrypt_output,16);
  printf("\r\n");
  printf("Server respones: ");


  if (isCard)
  {
    if (decrypt_output[0] != allow[0])
    {

      printf("FAIL");
      printf("\r\n");
      turnOnRedLight();
      ThisThread::sleep_for(2s);
      return false;
    }
    else
    {
      printf("SUCCESS");
      printf("\r\n");
      turnOnGreenLight();
      ThisThread::sleep_for(2s);
      return true;

    }
    ThisThread::sleep_for(1s);
  }
  else
  {
    if (decrypt_output[0] != allow[0])
    {
```

```
      printf("UNKNOWN DEVICE ID");
      printf("\r\n");
      return false;

    }
    else
    {
      printf("KNOWN DEVICE ID");
      printf("\r\n");
      return true;


    }
  }
}

int sendMessageToServer(TCPSocket *socket, struct message msg)
{
  char dev_type[] = "#";
  char card_type[] = "@";
  char hello_msg_type[] = "!";
  char encrypted_aes_type[] = "^";
  // Send the unique id of the RFID card to server
  int sent_bytes = (*socket).send(msg.payload, msg.length);
  printf("sent bytes are: %d \n", sent_bytes);
  if (sent_bytes <= 0)
  {
    return sent_bytes;
  }
  if (msg.type == dev_type[0])
  {
    printf("ENCRYPTED DEVICE UID: ");
    print_id(msg.payload, msg.length);
  }
  else if (msg.type == card_type[0])
  {
    printf("ENCRYPTED CARD UID: ");
    print_id(msg.payload, msg.length);
  }
  else if (msg.type == hello_msg_type[0])
  {
    printf("client sends:%s", msg.payload);
    printf("\n\r");
  }
  else if (msg.type == encrypted_aes_type[0])
  {
    printf("----ENCRYPTED AES KEY----\n");
    print_id(msg.payload, msg.length);
    printf("\n\r");
  }
  printf("\r\n");
  return sent_bytes;
}

bool receiveResponseFromServer(TCPSocket *socket, int bufferLength,
int isCard, unsigned char aes_key[])
{

  // Recieve a simple response and print out the response line
  char rbuffer[bufferLength];
```

```
  memset(rbuffer, 0, bufferLength); // clear the previous message
  struct message response_message;
  int rcount = (*socket).recv(rbuffer, sizeof rbuffer);

  response_message.type = rbuffer[0];
  response_message.length = (sizeof rbuffer);
  response_message.payload = rbuffer;

  printf("recv \n");
    for (int i = 0; i < 16; ++i) {
        printf("%X ", rbuffer[i]);
    }
  printf("\r\n");
  return decryptMessage(rbuffer, aes_key, isCard);

  //        printf("recv        %d        [%.*s]\n",        rcount,
strstr(response_message.payload, "\r\n") - response_message.payload,
response_message.payload);
}

int checkIfServerIsDown(int scount)
{

  if (scount <= 0)
  {

    LedBlue = 1;
    LedGreen = 0;
    LedRed = 0; // on
    printf("\r\n");
    printf("Server is down.Please wait!\n");
    printf("\r\n");
    ThisThread::sleep_for(3s);
    return 1;
  }
  return 0;
}

void encryptMessage(char plainText[], unsigned char aes_key[], char
encrypt_output[], int isCard)
{

  mbedtls_aes_context aes;
  mbedtls_aes_init(&aes);
  mbedtls_aes_setkey_enc(&aes, aes_key, 256); // 32 bytes key
  // cau_aes_set_key(aes_key, 256, (unsigned char *)60);

  unsigned char iv[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0e};
  unsigned char output_buffer[16];
  size_t INPUT_LENGTH = 16;
  mbedtls_aes_crypt_cbc(&aes, MBEDTLS_AES_ENCRYPT, INPUT_LENGTH, iv,
(const unsigned char *)plainText, output_buffer);
  //   mbedtls_aes_crypt_ecb(   &aes,   MBEDTLS_AES_ENCRYPT,   (const
unsigned char*)plainText, output_buffer);

  mbedtls_aes_free(&aes);

  // printf("-----aes encrypted message---- \n");
  // print_array(output_buffer,16);
  memmove(encrypt_output + 1, output_buffer, 16); // insert type of
```

```
message at buffer
  if (isCard == 0)
  {
    encrypt_output[0] = '#';
  }
  else
  {
    encrypt_output[0] = '@';
  }
}

void generateAndEncryptAesKey(unsigned char aes_key[], char
public_key[], size_t public_key_len, size_t aes_key_len, char
ecryptedAesKey[])
{
  mbedtls_ctr_drbg_context ctr_drbg;
  mbedtls_entropy_context entropy;
  // unsigned char key[32];

  char *pers = "aes generate key";
  int ret;

  mbedtls_entropy_init(&entropy);

  mbedtls_ctr_drbg_init(&ctr_drbg);

  if ((ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func,
&entropy,
                                   (unsigned char *)pers,
strlen(pers))) != 0)
  {
    printf(" failed\n ! mbedtls_ctr_drbg_init returned -0x%04x\n", -
ret);
  }

  if ((ret = mbedtls_ctr_drbg_random(&ctr_drbg, aes_key, 32)) != 0)
  {
    printf(" failed\n ! mbedtls_ctr_drbg_random returned -0x%04x\n",
-ret);
  }

  printf("\r\n");
  printf("----AES KEY---- \n");
  print_array(aes_key, 32);
  printf("\r\n");

  mbedtls_pk_context pk;

  mbedtls_pk_init(&pk);
  fflush(stdout);

  if ((ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func,
&entropy,
                                   (unsigned char *)pers,
strlen(pers))) != 0)
  {
    printf(" failed\n ! mbedtls_ctr_drbg_init returned -0x%04x\n", -
ret);
  }

  fflush(stdout);
```

```
  // public_key_len+1 because we use public key in pem format and it
is required from the function
  if   ((ret   =   mbedtls_pk_parse_public_key(&pk,   (unsigned   char
*)public_key, public_key_len + 1)) != 0)
  {
    printf("  failed\n    !  mbedtls_pk_parse_public_key  returned  -
0x%04x\n", -ret);
  }
  int buf_size = 128;
  unsigned char buf[128];
  size_t olen = 0;

  printf("Generating the encrypted aes key... \n");
  if  ((ret  =  mbedtls_pk_encrypt(&pk,  aes_key,  aes_key_len,  buf,
&olen, sizeof(buf), mbedtls_ctr_drbg_random, &ctr_drbg)) != 0)
  {
    printf(" failed\n   ! mbedtls_pk_encrypt  returned -0x%04x\n", -
ret);
  }

  mbedtls_pk_free(&pk);
  mbedtls_entropy_free(&entropy);
  mbedtls_ctr_drbg_free(&ctr_drbg);

  memmove(ecryptedAesKey + 1, buf, 128); // insert "^" at buf
  ecryptedAesKey[0] = '^';
}

void   findTagUniqueIDAndPost(int   communication_failed,   TCPSocket
*sock, unsigned char aes_key[])
{

  char card_id[5] = "";
  LedBlue = 1;
  printf("\r\n");
  printf("Scan your tag...\n");
  printf("\r\n");

  // Init. RC522 Chip
  RfChip.PCD_Init();

  while (communication_failed == 0)
  {

    // Look for new cards
    if (!RfChip.PICC_IsNewCardPresent())
    {
      LedGreen = 1;
      LedRed = 1;
      LedBlue = !LedBlue;
      ThisThread::sleep_for(100);
      continue;
    }

    // Select one of the cards
    if (!RfChip.PICC_ReadCardSerial())
    {
      LedGreen = 1;
      LedRed = 1;
      LedBlue = !LedBlue;
      ThisThread::sleep_for(100);
```

```
      continue;
    }

    LedBlue = 1;

    // Print Card UID
    for (uint8_t i = 0; i < RfChip.uid.size; i++)
    {
      card_id[i] = RfChip.uid.uidByte[i];
      // printf(" %X", RfChip.uid.uidByte[i]);
    }

    ThisThread::sleep_for(100);
    printf("\r\n");
    printf("----- CARD ID----- \n");
    print_byte_array(card_id, 4);
    char encrypted_card_id[17];
    encryptMessage(card_id, aes_key, encrypted_card_id, 1);

    struct message card_message;

    card_message.type = encrypted_card_id[0];
    card_message.length = (sizeof encrypted_card_id);
    card_message.payload = encrypted_card_id;

    int scount = sendMessageToServer(sock, card_message);

    // if server is down break and try again
    communication_failed = checkIfServerIsDown(scount);
    if (communication_failed)
    {
      break;
    }

    turnOffAllLights();
    receiveResponseFromServer(sock, 25, 1, aes_key);
  }
}
//
int askForPublicKey(TCPSocket *socket)
{

  struct message hello_message;
  char hello_msg[] = "!";

  hello_message.type = hello_msg[0];
  hello_message.length = (sizeof hello_msg);
  hello_message.payload = hello_msg;

  int rec = sendMessageToServer(socket, hello_message);
  int communication_failed = checkIfServerIsDown(rec);

  return communication_failed;
}

void receivePublicKey(TCPSocket *socket, char public_key[], int n)
{
  char rbuffer[n];
  memset(rbuffer, 0, n); // clear the previous message
  // struct message response_message;
  int rcount = (*socket).recv(rbuffer, n);
```

```
  // printf("length of public key is: %d \n", rcount);

  memcpy(public_key, rbuffer, n);
  printf("\r\n");
  printf("----PUBLIC KEY---- \n");
  for (int i = 0; i < n; ++i)
  {
    printf("%X ", public_key[i]);
  }
  printf("\r\n");
  // printf("copy [%.*s]\n", strstr(public_key, "\r\n") - public_key,
public_key);
}

bool signalSentSuccessfully(TCPSocket *socket, char signal[])
{
  struct message signalAsStruct;

  signalAsStruct.type = signal[0];
  signalAsStruct.length = (sizeof signal);
  signalAsStruct.payload = signal;
  int bytesSent = sendMessageToServer(socket, signalAsStruct);
  if (checkIfServerIsDown(bytesSent) == 1)
  {
    return false;
  }
  return true;
}

int readNumber(TCPSocket *socket)
{
  char bytesBuffer[4];
  char numberBuffer[9];
  memset(bytesBuffer, 0, 4);
  memset(numberBuffer, 0, 9);
  (*socket).recv(bytesBuffer, 4);
  for (int i = 0; i < 4; ++i)
  {
    sprintf(&numberBuffer[i * 2], "%02X", bytesBuffer[i]);
  }

  int number = (int)strtol(numberBuffer, NULL, 16);
  printf("number is : %d\n",number);
  memset(bytesBuffer, 0, 4);
  memset(numberBuffer, 0, 9);
  return number;
}

int askForP(TCPSocket *socket)
{
  if (!signalSentSuccessfully(socket, "P"))
  {
    return FAIL_TO_RECEIVE_DIFF_HELLMAN_PARAM;
  }

  return readNumber(socket);
}

int askForG(TCPSocket *socket)
{
```

```
  if (!signalSentSuccessfully(socket, "G"))
  {
    return FAIL_TO_RECEIVE_DIFF_HELLMAN_PARAM;
  }

  return readNumber(socket);
}

int askForB(TCPSocket *socket)
{
  if (!signalSentSuccessfully(socket, "B"))
  {
    return FAIL_TO_RECEIVE_DIFF_HELLMAN_PARAM;
  }

  return readNumber(socket);
}

int askForKeyExchangeMethod(TCPSocket *socket)
{
  if (!signalSentSuccessfully(socket, "*"))
  {
    return FAIL_TO_RECEIVE_DIFF_HELLMAN_PARAM;
  }

  return readNumber(socket);
}


bool sendA(TCPSocket *socket, int number)
{
  struct message shared_number;
  char shared_msg[4];
  sprintf(shared_msg, "&%01d", number);
  // printf("%s\n", shared_msg);

  shared_number.type = shared_msg[0];
  shared_number.length = (sizeof shared_msg);
  shared_number.payload = shared_msg;
  int bytesSent = sendMessageToServer(socket, shared_number);
  if (checkIfServerIsDown(bytesSent))
  {
    return false;
  }
  return true;
}

int bringUpEthernetConnection(TCPSocket *socket)
{

  net.connect();

  // Show the network address
  SocketAddress socketAddr;
  net.get_ip_address(&socketAddr);
  printf("IP    address:    %s\n",    socketAddr.get_ip_address()    ?
socketAddr.get_ip_address() : "None"); // the IP address of frdm
  if (socketAddr.get_ip_address() == nullptr)
  {
    return 1;
  }
```

```
  // Open a socket on the network interface, and create a TCP
connection

  (*socket).open(&net);
  // 192.168.2.17  192.168.1.3

  net.gethostbyname("192.168.1.2", &socketAddr);
  socketAddr.set_port(8080);
  (*socket).connect(socketAddr);
  return 0;
}

int diffHellmanFormula(int P, int G, int secreNumber)
{
  int gPowered = pow(double(G), double(secreNumber));
  return (gPowered % P);
}

void putSecretToAesKey(unsigned char bytesBuffer[], int secret)
{
  // char bytesBuffer[16];
  memset(bytesBuffer, 0, 8);

  // bytesBuffer[12] = (secret >> 24) & 0xFF;
  // bytesBuffer[13] = (secret >> 16) & 0xFF;
  // bytesBuffer[14] = (secret >> 8) & 0xFF;
  // bytesBuffer[15] = secret & 0xFF;
  bytesBuffer[28] = (secret >> 24) & 0xFF;
  bytesBuffer[29] = (secret >> 16) & 0xFF;
  bytesBuffer[30] = (secret >> 8) & 0xFF;
  bytesBuffer[31] = secret & 0xFF;
}



int main(int argc, char **argv)
{
  //TCPSocket socket;
  turnOnBlueLight();
  int communication_failed = 0;


  char device_id[16];
  findUniqueDeviceId(device_id);


  while (1)
  {
    printf("\r\n");
    printf("Door Lock System\n");
    printf("Connecting to server...\n");
    TCPSocket socket; // it is must here



    // if connection fail try again
    if (bringUpEthernetConnection(&socket))
    {
      continue;
    }
```

```
    int diffie_hellman=askForKeyExchangeMethod(&socket);

    printf("DIFFIE HELLMAN: %d \n",diffie_hellman );


    unsigned char aes_key[32];
    size_t aes_key_size = sizeof aes_key / sizeof aes_key[0];

    if (diffie_hellman)
    {
      printf("\r\n");
      printf("DIFFIE HELLMAN AND AES HYBRID CRYPTOGRAPHY\n");
      printf("\r\n");
      const int P = askForP(&socket);
      if (P == FAIL_TO_RECEIVE_DIFF_HELLMAN_PARAM)
      {
        printf("Could not retrieve Diff Hellman P parameter. Will
retry to reconnect in 2s...\n");
        ThisThread::sleep_for(2s);
        continue;
      }

      const int G = askForG(&socket);
      if (G == FAIL_TO_RECEIVE_DIFF_HELLMAN_PARAM)
      {
        printf("Could not retrieve Diff Hellman G parameter. Will
retry to reconnect in 2s...\n");
        ThisThread::sleep_for(2s);
        continue;

      }

      srand((unsigned int)**main + (unsigned int)&argc + (unsigned
int)time(NULL));
      const int secretNumber = rand() % 5;
      const int A = diffHellmanFormula(P, G, secretNumber);
      if (!sendA(&socket, A))
      {
        printf("Could not send Diff Hellman A parameter. Will retry
to reconnect in 2s...\n");
        ThisThread::sleep_for(2s);
        continue;
      }

      const int B = askForB(&socket);
      printf("Diff Hellman Numbers: P: %d, G: %d, privateNumber: %d,
A: %d, B: %d\n", P, G, secretNumber, A, B);
      const    int    sharedSecret    =    diffHellmanFormula(P,    B,
secretNumber);
      printf("---SECRET NUMBER---:%d\n", sharedSecret);
      printf("\r\n");
      //unsigned char aesKey[32];
      memset(aes_key, 0, aes_key_size); // clear the previous message
      putSecretToAesKey(aes_key, sharedSecret);
      for (int i = 0; i < 32; i++)
      {
        printf("%X ", aes_key[i]);
      }
      printf("\r\n");
```

```
    }

    else
    {
      printf("\r\n");
      printf("RSA AND AES HYBRID CRYPTOGRAPHY\n");
      printf("\r\n");
      //------------CLIENT ASK FOR PUBLIC KEY------------

      if (askForPublicKey(&socket))
      {
        continue;
      }

      //----------CLIENT RECIEVE PUBLIC KEY--------------------
      int public_key_length = 271;
      char public_key[public_key_length];
      size_t   public_key_size   =   sizeof   public_key   /   sizeof
public_key[0];

      receivePublicKey(&socket, public_key, public_key_length);

      //----------GENERATE AND ENCRYPT AES KEY--------------------



      char ecryptedAesKey[129];
      generateAndEncryptAesKey(aes_key,  public_key,  public_key_size,
aes_key_size, ecryptedAesKey);
      printf("------------------------------------ \n");

      struct message encryptedAesKey_message;

      encryptedAesKey_message.type = ecryptedAesKey[0];
      encryptedAesKey_message.length = (sizeof ecryptedAesKey);
      encryptedAesKey_message.payload = ecryptedAesKey;

      int s = sendMessageToServer(&socket, encryptedAesKey_message);
      communication_failed = checkIfServerIsDown(s);
      if (communication_failed)
      {
       continue;
      }

    }

    printf("-----AFTER KEY EXCHANGE-----\n" );

    for (int i = 0; i < 32; i++)
    {
      printf("%X ", aes_key[i]);
    }
    printf("\r\n");


    //----------CLIENT SEND DEVICE ID--------------------
      char encrypted_device_id[17];
      printf("----- DEVICE ID----- \n");
      print_byte_array(device_id, 16);
      encryptMessage(device_id, aes_key, encrypted_device_id, 0);
```

```
      struct message device_message;

      device_message.type = encrypted_device_id[0];
      device_message.length = (sizeof encrypted_device_id);
      device_message.payload = encrypted_device_id;

      int s = sendMessageToServer(&socket, device_message);

      if (checkIfServerIsDown(s))
      {
        continue;
      }

      bool  known_device=receiveResponseFromServer(&socket,  34,  0,
aes_key);
      if(!known_device){
        break;
      }

      //---------------SCAN TAGS-----------------------

      findTagUniqueIDAndPost(communication_failed, &socket, aes_key);

    // Close the socket to return its memory and bring down the
network interface
    socket.close();

    // Bring down the ethernet interface
    net.disconnect();
  }

  printf("Sorry but this device is unknown.\n");
}
```

**server.py**

```
import socket, select
import binascii
import sqlite3
import sys
import traceback
import random
import re
from _thread import *
from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import PKCS1_v1_5
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
from Crypto.Cipher import AES
from math import gcd


#   https://www.geeksforgeeks.org/find-the-number-of-primitive-roots-
modulo-prime/
def findPrimitiveRoot(p):
```

```
    result = 1
    for i in range(2, p, 1):
        if (gcd(i, p) == 1):
            return i


#      https://www.delftstack.com/howto/python/python-generate-prime-
number/
def primesInRange(x, y):
    prime_list = []
    for n in range(x, y):
        isPrime = True
        for num in range(2, n):
            if n % num == 0:
                isPrime = False
        if isPrime:
            prime_list.append(n)
    return prime_list


def search_for_pair_in_database(device_id, card_id):
    # connect with database
    #     conn     =     sqlite3.connect(r"C:\Users\Dora\Desktop\Start
Diplomatiki\DoorLock.db")
    conn = sqlite3.connect(r"DoorLock.db")
    print("pair: ", device_id, "-", card_id)

    cur = conn.cursor()
    cur.execute("SELECT DoorID FROM `User` WHERE ID = ?", [card_id])

    result = cur.fetchone()

    if (result == None):
        text = b'0'  # disallow
        print("PAIR DOES NOT EXISTS.")
        out = aes_encryption(aes_key, text)
        return out
    else:
        for row in result:
            if (row == device_id):
                text = b'1'  # allow
                print("PAIR EXISTS.")
                out = aes_encryption(aes_key, text)
                return out


def search_for_device_id_in_database(device_id):
    # connect with database
    conn = sqlite3.connect(r"DoorLock.db")

    cur = conn.cursor()

    cur.execute("SELECT ID FROM `Door` WHERE ID = ?", [device_id])

    result = cur.fetchone()
    if (result == None):
        text = b'0'  # "Unknown device."
        out = aes_encryption(aes_key, text)
        return out
    else:
        for row in result:
```

```
            if (row == device_id):
                text = b'1'  # known device id
                out = aes_encryption(aes_key, text)
                return out


def aes_encryption(aes_key, text):
    len_of_text = len(text)

    bytes_val = len_of_text.to_bytes(1, 'big')
    # print(bytes_val)

    if (len(text) < 16):
        text = text.ljust(16, b'0')
    iv                                                                  =
b"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0e"
    aes = AES.new(aes_key, AES.MODE_CBC, iv)  # Create an aes object
    # AES. MODE_ The CBC representation pattern is the CBC pattern
    en_text = aes.encrypt(text)
    #  print("Ciphertext:", en_text)   # Encrypted plaintext, bytes
type
    # print(en_text.hex())
    return en_text


def aes_decryption(aes_key, encrypted_data):
    iv                                                                  =
b"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0e"
    aes = AES.new(aes_key, AES.MODE_CBC, iv)   # Decryption in CBC
mode requires re creating an aes object
    den_text = aes.decrypt(encrypted_data)
    print("Plaintext:", den_text.hex())
    output = den_text.hex()
    return output


def generate_rsa_keys():
    new_key = RSA.generate(1024)

    private_key = new_key.exportKey("PEM")
    public_key = new_key.publickey().exportKey("PEM")
    print(public_key)

    fd = open("private_key.pem", "wb")
    fd.write(private_key)
    fd.close()

    fd = open("public_key.pem", "wb")
    fd.write(public_key)
    fd.close()
    return public_key


def stringToBytes(s):
    b = bytearray()
    b = bytearray()
    b.extend(map(ord, s))
    return b


def diffHellmanFormula(p, g, private_number):
```

```
    g_powered = pow(g, private_number)
    calculated_number = g_powered % p
    return calculated_number


def numberAsBytes(num):
    return num.to_bytes(4, 'big')




def multi_threaded_client(connection):
    global aes_key, P, G, A, secret_key_bytes


    while True:
        data = connection.recv(2048)
        # print("len of data:", len(data))
        dataAsHex = data.hex()
        request = data[0:1].decode("utf-8")

        if request == "*":
            key_exchange_method  =  int(input("Choose  key  exchange
method. Press 1 for Diffie Hellman and 0 for RSA:"))
            print("message is:", data[0:1].decode("utf-8"))
            print(key_exchange_method, type(key_exchange_method))
            connection.send(numberAsBytes(key_exchange_method))

        elif request == "!":
            print("message is:", data.decode("utf-8"))
            public_key = generate_rsa_keys()
            connection.send(public_key)

        elif request == "^":
            print("-------------------------------------------------
----------")
            print("Encrypted  AES  key",  data[0:1].decode("utf-8")  +
dataAsHex[2:len(dataAsHex)])
            key = RSA.import_key(open('private_key.pem').read())
            r = Random.new().read(128)
            cipher = PKCS1_v1_5.new(key)
            aes_key = cipher.decrypt(data[1:len(data)], r)
            print("----AES KEY----")
            print(aes_key.hex())
            print("-------------------------------------------------
----------")
        elif request == "P":
            private_number = random.randint(0, 20)
            print(private_number)
            prime_list = primesInRange(5, 250)
            P = random.choice(prime_list)
            print("privateNumber is: ", private_number)
            print("P is: ", P)
            connection.send(numberAsBytes(P))
        elif request == "G":
            G = findPrimitiveRoot(P - 1)
            print("G is: ", G)
            connection.send(numberAsBytes(G))
        elif request == "&":
```

```
            decodedData = data.decode("utf-8")
            cropped_data = decodedData[1:len(decodedData)]
            cropped_data = cropped_data.strip()
            cropped_data = cropped_data + ""
            cropped_data = re.sub("[^0-9]", "", cropped_data)
            A = int(float(cropped_data))
            print("A is: ", A)

        elif request == "B":
            B = diffHellmanFormula(P, G, private_number)
            print("B is: ", B)
            connection.send(numberAsBytes(B))
            secret_key = diffHellmanFormula(P, A, private_number)
            aes_key = secret_key.to_bytes(32, 'big')
            print("---SECRET KEY----")
            print(secret_key)
            print("----AES KEY in hex---- ")
            print(aes_key.hex())
            # print(aes_key_hex)

            print("--------------------------------------------------
----------")
            # encrypted = aes_encryption(secret_key_bytes, "hello")
            # print("ENCRTPYRTTED:  ")
            # print(encrypted)
            # aes_decryption(secret_key_bytes,encrypted)

        elif request == "#":
            print("Encrypted Device ID:", data[0:1].decode("utf-8") +
dataAsHex[2:len(dataAsHex)])
            device_id = dataAsHex[2:len(dataAsHex)]

            device_id = aes_decryption(aes_key, data[1:len(data)])
            access_message                                          =
search_for_device_id_in_database(device_id)
            connection.send(access_message)

        elif request == "@":
            print("Encrypted  Card  ID:",  data[0:1].decode("utf-8")  +
dataAsHex[2:len(dataAsHex)])

            card_id = aes_decryption(aes_key, data[1:len(data)])
            print(card_id[:8])
            access_message  =  search_for_pair_in_database(device_id,
card_id[0:8])
            connection.send(access_message)
        else:
            connection.send(stringToBytes("SORRY"))
    connection.close()

def main():
    ServerSideSocket = socket.socket()
    host = '127.0.0.1'
    port = 8080
    ThreadCount = 0
    try:
        ServerSideSocket.bind(("0.0.0.0", port))
    except socket.error as e:
        print(str(e))
    print('Socket is listening..')
    ServerSideSocket.listen(5)
```

```
    while True:
        Client, address = ServerSideSocket.accept()
        print('Connected to: ' + address[0] + ':' + str(address[1]))
        start_new_thread(multi_threaded_client, (Client, ))
        ThreadCount += 1
        print('Thread Number: ' + str(ThreadCount))
    ServerSideSocket.close()



if __name__ == "__main__":
    main()
```

ПАРАРТНМА В

```
/*
 * Copyright (c) 2015, Freescale Semiconductor, Inc.
 * Copyright 2016-2021 NXP
 * All rights reserved.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */

/*************************************************************************
***********
 * Includes

*************************************************************************
*********/
#include <stdlib.h>
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "board.h"

#include "fsl_mmcau.h"

/*************************************************************************
***********
 * Definitions

*************************************************************************
*********/

#define CORE_CLK_FREQ CLOCK_GetFreq(kCLOCK_CoreSysClk)
/* Number of cycles for througput measurement. One data buffer of
certain size if processed this times. */
#define CYCLES_FOR_THROUGHPUT 10000
#define CYCLES_FOR_PASSRATE   1024

/*AES specific*/
#define AES128           128
#define AES128_KEY_SIZE  16
```

```
#define AES_BLOCK_LENGTH 16

#define AES192          192
#define AES192_KEY_SIZE 24

#define AES256          256
#define AES256_KEY_SIZE 32


/*length of AES & DES encrypted data array*/
#define OUTPUT_ARRAY_LEN 512


/*size of one crypto block*/
#define CRYPTO_BLOCK_LENGTH 64
/*size of padded hash input string*/
#define TEST_LENGTH 64

/*MMCAU result codes*/
#define MMCAU_OK    0
#define MMCAU_ERROR -1

/*********************************************************************
***********
 * Variables

*********************************************************************
*********/
/*16 bytes key: "ultrapassword123"*/
const uint8_t g_aesKey128[AES128_KEY_SIZE] = {0x75, 0x6c, 0x74, 0x72,
0x61, 0x70, 0x61, 0x73,
                                              0x73, 0x77, 0x6f, 0x72,
0x64, 0x31, 0x32, 0x33};
/*16 bytes key: "ultrapassword123ultrapas"*/
const uint8_t g_aesKey192[AES192_KEY_SIZE] = {0x75, 0x6c, 0x74, 0x72,
0x61, 0x70, 0x61, 0x73, 0x73, 0x77, 0x6f, 0x72,
                                              0x64, 0x31, 0x32, 0x33,
0x75, 0x6c, 0x74, 0x72, 0x61, 0x70, 0x61, 0x73};

/*32 bytes key: "ultrapassword123ultrapassword123"*/
const uint8_t g_aesKey256[AES256_KEY_SIZE] = {0x75, 0x6c, 0x74, 0x72,
0x61, 0x70, 0x61, 0x73, 0x73, 0x77, 0x6f,
                                              0x72, 0x64, 0x31, 0x32,
0x33, 0x75, 0x6c, 0x74, 0x72, 0x61, 0x70,
                                              0x61, 0x73, 0x73, 0x77,
0x6f, 0x72, 0x64, 0x31, 0x32, 0x33};

/*initialization vector: 16 bytes: "mysecretpassword"*/
const uint8_t g_aesIV[AES_BLOCK_LENGTH] = {0x6d, 0x79, 0x73, 0x65,
0x63, 0x72, 0x65, 0x74,
                                           0x70, 0x61, 0x73, 0x73,
0x77, 0x6f, 0x72, 0x64};



static const uint8_t g_testFull[] ="HELLOOOOOOOOOOO";
float set_up_key_time;


static uint8_t g_output[OUTPUT_ARRAY_LEN];
static uint8_t g_result[OUTPUT_ARRAY_LEN];
```

```
static volatile uint32_t g_msCount = 0;
static volatile bool g_irq_random  = false;

/*******************************************************************
***********
 * Prototypes

*******************************************************************
*********/
static void mmcau_print_msg(const uint8_t *data, uint32_t length);
static void mmcau_example_task(void);

/*******************************************************************
***********
 * Code

*******************************************************************
*********/

/*!
 * @brief counter since last POR/reset.
 */
void SysTick_Handler(void)
{
    g_msCount++;

}

/*!
 * @brief SysTick period configuration and interrupt enable.
 */
static uint32_t time_config(bool random)
{

    {
        g_irq_random = false;
        /*  call  CMSIS  SysTick  function.  It  enables  the  SysTick
interrupt at low priority */
        return SysTick_Config(CORE_CLK_FREQ / 1000); /* 1 ms period
*/
    }
}

/*!
 * @brief Get milliseconds since last POR/reset.
 */
static float time_get_ms(void)
{
    uint32_t currMsCount;
    uint32_t currTick;
    uint32_t loadTick;

    do
    {
        currMsCount = g_msCount;
        currTick    = SysTick->VAL;
    } while (currMsCount != g_msCount);

    loadTick = CORE_CLK_FREQ / 1000;
    return (float)currMsCount + ((float)loadTick - (float)currTick) /
```

```
(float)loadTick;
}

/*!
 * @brief Get througput in MB/s
 * @param elapsedMs Time interval in milliseconds.
 *  @param numBytes Number of bytes processed within the given
interval.
 * @return Throughput in MB/s.
 */
static float mmcau_get_throughput(float elapsedMs, size_t numBytes)
{

    return ((float)(1000 * numBytes / 1024 / 1024) / elapsedMs);
}

float get_cycles_per_byte(float throughput)
{
  return ((float)(120*pow(10,6))/(throughput*pow(1024,2)));
}

/*!
 * @brief Function mmcau_print_msg prints data bytes into console.
 */
static void mmcau_print_msg(const uint8_t *data, uint32_t length)
{
    uint32_t i;

    PRINTF("          ");
    for (i = 0; i < length; i++)
    {
        PUTCHAR(data[i]);
        if (data[i] == ',')
        {
            PRINTF("\r\n          ");
        }
    }
    PRINTF("\r\n");
}

/*
 * mmcau_encrypt_aes_cbc: AES encryption function
 *
 * Parameters:
 *   [in] key: key of 8 bytes
 *   [in] mode: 128, 192 or 256 AES mode
 *   [in] inputData: pointer to in data
 *   [out] outputData: pointer to out data
 *     [in]  dataLength:  number  of  bytes  of  input  data.  Must  be
divisible by 8 (DES block)
 *   [in] initVector: initVector to use during xor
 * Return:
 *   0 if OK, otherwise error
 *
 */
static int mmcau_encrypt_aes_cbc(const uint8_t *key,
                                 uint32_t mode,
                                 const uint8_t *inputData,
                                 uint8_t *outputData,
                                 uint16_t dataLength,
                                 const uint8_t *initVector)
```

```
{
    uint8_t i;
    uint16_t blocks;
    uint16_t rounds;
    uint8_t tempBlock[AES_BLOCK_LENGTH];
    uint8_t tempIV[AES_BLOCK_LENGTH];
    float timeBefore;
    float timeAfter;
    float totalTime;
    float elapsedTime;
    /*
     * AES128 needs 44 longwords for expansion
     * AES192 needs 52 longwords for expansion
     * AES256 needs 60 longwords for expansion
     *    Reserving 60 longwords as the max space
     */
    uint32_t keyLen;
    uint8_t keyExpansion[60 * 4];

    /*validate NULL for key, inputData, outputData or initVector*/
    if ((key == NULL) || (inputData == NULL) || (outputData == NULL)
|| (initVector == NULL))
    {
        return MMCAU_ERROR; /*wrong pointer*/
    }

    /*validate AES mode*/
    if ((mode != 128u) && (mode != 192u) && (mode != 256u))
    {
        return MMCAU_ERROR; /*wrong AES mode*/
    }

    /*validate data length*/
    if (dataLength % AES_BLOCK_LENGTH)
    {

        return MMCAU_ERROR; /*wrong length*/
    }

    /*calculate number of AES rounds*/
    if (mode == 128u)
    {
        rounds = 10u;
        keyLen = 16u;
    }
    else if (mode == 192u)
    {
        rounds = 12u;
        keyLen = 24u;
    }
    else /*AES256*/
    {
        rounds = 14u;
        keyLen = 32u;
    }

    /*expand AES key*/
    //UNCOMMENT IF YOU WANT TO MEASURE SET UP KEY TIME
    //timeBefore=time_get_ms();
    MMCAU_AES_SetKey(key, keyLen, keyExpansion);
    //timeAfter=time_get_ms();
```

```
    //elapsedTime=timeAfter-timeBefore;
    //set_up_key_time+=elapsedTime*1000;
    //PRINTF("set    up    key    time    encryption:    %f    \r\n",
elapsedTime*1000);

    /*execute AES in CBC mode*/
    /*http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation*/

    /*get number of blocks*/
    blocks = dataLength / AES_BLOCK_LENGTH;

    /*copy init vector to temp storage*/
    memcpy(tempIV, initVector, AES_BLOCK_LENGTH);

    do
    {
        /*copy to temp storage*/
        memcpy(tempBlock, inputData, AES_BLOCK_LENGTH);

        /*xor for CBC*/
        for (i = 0; i < AES_BLOCK_LENGTH; i++)
        {
            tempBlock[i] ^= tempIV[i];
        }

        /*FSL: single AES round*/
        MMCAU_AES_EncryptEcb(tempBlock,     keyExpansion,     rounds,
outputData);

        /*update initVector for next 3DES round*/
        memcpy((void *)tempIV, (void *)outputData, AES_BLOCK_LENGTH);

        /*adjust pointers for next 3DES round*/
        inputData += AES_BLOCK_LENGTH;
        outputData += AES_BLOCK_LENGTH;

    } while (--blocks);

    return MMCAU_OK;
}

/*
 * mmcau_decrypt_aes_cbc: AES decryption function
 *
 * Parameters:
 *   [in] key: key of 8 bytes
 *   [in] mode: 128, 192 or 256 AES mode
 *   [in] inputData: pointer to in data
 *   [out] outputData: pointer to out data
 *     [in]  dataLength:  number  of  bytes  of  input  data.  Must  be
divisible by 8 (DES block)
 *   [in] initVector: initVector to use during xor
 * Return:
 *   0 if OK, otherwise error
 *
 */
static int mmcau_decrypt_aes_cbc(const uint8_t *key,
                                 uint32_t mode,
                                 const uint8_t *inputData,
                                 uint8_t *outputData,
                                 uint16_t dataLength,
```

```
                                    const uint8_t *initVector)
{
    uint8_t i;
    uint16_t blocks;
    uint16_t rounds;
    uint8_t tempBlock[AES_BLOCK_LENGTH];
    uint8_t tempIV[AES_BLOCK_LENGTH];
    float timeBefore;
    float timeAfter;
    float elapsedTime;
    /*
     * AES128 needs 44 longwords for expansion
     * AES192 needs 52 longwords for expansion
     * AES256 needs 60 longwords for expansion
     *    Reserving 60 longwords as the max space
     */
    uint8_t keyExpansion[60 * 4];
    uint32_t keyLen;

    /*validate NULL for key, inputData, outputData or initVector*/
    if ((key == NULL) || (inputData == NULL) || (outputData == NULL)
|| (initVector == NULL))
    {
        return MMCAU_ERROR; /*wrong pointer*/
    }

    /*validate AES mode*/
    if ((mode != 128u) && (mode != 192u) && (mode != 256u))
    {
        return MMCAU_ERROR; /*wrong AES mode*/
    }

    /*validate data length*/
    if (dataLength % AES_BLOCK_LENGTH)
    {
        return MMCAU_ERROR; /*wrong length*/
    }

    /*calculate number of AES rounds*/
    if (mode == 128u)
    {
        rounds = 10u;
        keyLen = 16u;
    }
    else if (mode == 192u)
    {
        rounds = 12u;
        keyLen = 24u;
    }
    else /*AES256*/
    {
        rounds = 14u;
        keyLen = 32u;
    }

    /*expand AES key*/
    //timeBefore=time_get_ms();
    MMCAU_AES_SetKey(key, keyLen, keyExpansion);
    //timeAfter=time_get_ms();
    //elapsedTime=timeAfter-timeBefore;
    //set_up_key_time+=elapsedTime*1000;
```

```
    //PRINTF("set up key time decryption: %f\r\n", elapsedTime*1000);


    /*execute AES in CBC mode*/
    /*http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation*/

    /*get number of blocks*/
    blocks = dataLength / AES_BLOCK_LENGTH;

    /*copy init vector to temp storage*/
    memcpy(tempIV, initVector, AES_BLOCK_LENGTH);

    do
    {
        /*copy to temp storage*/
        memcpy(tempBlock, inputData, AES_BLOCK_LENGTH);

        /*FSL: single AES round*/
        MMCAU_AES_DecryptEcb(inputData,     keyExpansion,     rounds,
outputData);

        /*xor for CBC*/
        for (i = 0; i < AES_BLOCK_LENGTH; i++)
        {
            outputData[i] ^= tempIV[i];
        }

        /*update initVector for next AES round*/
        memcpy(tempIV, tempBlock, AES_BLOCK_LENGTH);

        /*adjust pointers for next 3DES round*/
        inputData += AES_BLOCK_LENGTH;
        outputData += AES_BLOCK_LENGTH;

    } while (--blocks);

    return MMCAU_OK;
}



/*!
 * @brief Function mmcau_example_task demonstrates use of mmcau
 *        encryption/decryption functions on short sample text.
 */
static void mmcau_example_task(void)
{
    uint32_t length;
    uint32_t blocks;
    uint8_t status;
    float timeBefore;
    float timeAfter;
    float timeBefore1;
    float timeAfter1;
    float total_time;
    float elapsed_time;
    float throughtput;
    int cycles;

    /* Print welcome string */
```

```
    PRINTF("............................ MMCAU    DRIVER    EXAMPLE
............................ \r\n\r\n");
    memset(&g_output[0], 0, OUTPUT_ARRAY_LEN);
    memset(&g_result[0], 0, OUTPUT_ARRAY_LEN);
    length = sizeof(g_testFull) - 1u;

    PRINTF("Testing input string: \r\n");
    mmcau_print_msg(&g_testFull[0], length);
    /* Format console output */
    PRINTF("\r\n");

    /************************************************/
    /******* FIRST PART USING AES-CBC method *********/
    /************************************************/
    PRINTF("---------------------------------- AES-128-CBC method --
-----------------------------------\r\n");

    /*   ENCRYPTION   */
    PRINTF("AES-256 CBC decryption of %d bytes.\r\n", length);



    /**********************MEASURE                                  POWER
CONSUMPTION*****************************/
//
//    while (1)
//    {
//
//            //status  =  mmcau_encrypt_aes_cbc(g_aesKey128,  AES128,
g_testFull, g_output, length, g_aesIV);
//          //status   =   mmcau_decrypt_aes_cbc(g_aesKey128,   AES128,
g_output, g_result, length, g_aesIV);
//          //status   =   mmcau_encrypt_aes_cbc(g_aesKey192,   AES192,
g_testFull, g_output, length, g_aesIV);
//          //status   =   mmcau_decrypt_aes_cbc(g_aesKey192,   AES192,
g_output, g_result, length, g_aesIV);
//          //status   =   mmcau_encrypt_aes_cbc(g_aesKey256,   AES256,
g_testFull, g_output, length, g_aesIV);
//          status   =   mmcau_decrypt_aes_cbc(g_aesKey256,   AES256,
g_output, g_result, length, g_aesIV);
//    }


    /*************** COUNT MILLISECONDS PER OPERATION  ************/

    /* Call AES_cbc encryption */
    //uint32_t keyLen;
    //uint8_t keyExpansion[60 * 4];
    //MMCAU_AES_SetKey(g_aesKey128, 16u, keyExpansion);
    cycles     = CYCLES_FOR_THROUGHPUT;
    timeBefore1 = time_get_ms();
    while (cycles)
    {
      //timeBefore1 = time_get_ms();
        status    =    mmcau_encrypt_aes_cbc(g_aesKey128,    AES128,
g_testFull, g_output, length, g_aesIV);
        //timeAfter1 = time_get_ms();
        //elapsed_time=timeAfter1 - timeBefore1;
        //total_time+=timeAfter1 - timeBefore1;
        //PRINTF("elapsed   time   %f   ms.\r\n\r\n",timeAfter1   -
timeBefore1);
```

```
        if (status != MMCAU_OK)
        {
            PRINTF("AES-128 CBC encryption failed !\r\n");
            return;
        }
        cycles--;
    }
    timeAfter1 = time_get_ms();
    elapsed_time=timeAfter1 - timeBefore1;

    PRINTF("ms              per              operation:              %f
ms.\r\n\r\n",(elapsed_time/CYCLES_FOR_THROUGHPUT));
    PRINTF("-----set    up    key    time    total    encrypt    %f
ms.\r\n\r\n",set_up_key_time/CYCLES_FOR_THROUGHPUT);
    set_up_key_time=0.0;



    /******************                              COUNT       THROUGHPUT
**********************/
    cycles     = CYCLES_FOR_THROUGHPUT;
    timeBefore = time_get_ms();
    while (cycles)
    {

        status    =    mmcau_encrypt_aes_cbc(g_aesKey128,    AES128,
g_testFull, g_output, length, g_aesIV);

        if (status != MMCAU_OK)
        {
            PRINTF("AES-128 CBC encryption failed !\r\n");
            return;
        }
        cycles--;
    }
    timeAfter = time_get_ms();

    PRINTF("Time  was  taken  in  encrypt  %f  ms.\r\n\r\n",timeAfter  -
timeBefore);
    set_up_key_time=0.0;



    /* Result message */
    PRINTF("AES-128 CBC encryption finished. Speed %f MB/s.\r\n\r\n",
           mmcau_get_throughput(timeAfter        -        timeBefore,
CYCLES_FOR_THROUGHPUT * length));


    throughtput=mmcau_get_throughput(timeAfter      -      timeBefore,
CYCLES_FOR_THROUGHPUT * length);

    printf("AES  CBC  encryption  cycles  per  byte  %.3f  cycles  per
byte.\r\n\r\n",
           get_cycles_per_byte(throughtput));



    /*   DECRYPTION   */
    PRINTF("AES-128 CBC Decryption of %d bytes.\r\n", length);
```

```
    /* Call AES_cbc decryption */
    set_up_key_time=0.0;
    cycles= CYCLES_FOR_THROUGHPUT;
    //MMCAU_AES_SetKey(g_aesKey128, 16u, keyExpansion);
    timeBefore1 = time_get_ms();
    while (cycles)
    {
      //timeBefore1 = time_get_ms();
        status = mmcau_decrypt_aes_cbc(g_aesKey128, AES128, g_output,
g_result, length, g_aesIV);
//        timeAfter1 = time_get_ms();
//        elapsed_time=timeAfter1 - timeBefore1;
//        total_time+=elapsed_time;
        if (status != MMCAU_OK)
        {
            PRINTF("AES-128 CBC decryption failed !\r\n");
            return;
        }
        cycles--;
    }
    timeAfter1 = time_get_ms();
    elapsed_time=timeAfter1 - timeBefore1;

    PRINTF("ms              per              operation:              %f
ms.\r\n\r\n",(elapsed_time/CYCLES_FOR_THROUGHPUT));
    PRINTF("-----set    up    key    time    total    decrypt    %f
ms.\r\n\r\n",set_up_key_time/CYCLES_FOR_THROUGHPUT);
    set_up_key_time=0.0;

    /* Result message */

    cycles     = CYCLES_FOR_THROUGHPUT;
    timeBefore = time_get_ms();
    set_up_key_time=0.0;
    while (cycles)
    {
      //timeBefore1 = time_get_ms();
        status = mmcau_decrypt_aes_cbc(g_aesKey128, AES128, g_output,
g_result, length, g_aesIV);
        //timeAfter1 = time_get_ms();
        //elapsed_time=timeAfter1 - timeBefore1;
        //total_time+=elapsed_time;
        if (status != MMCAU_OK)
        {
            PRINTF("AES-128 CBC decryption failed !\r\n");
            return;
        }
        cycles--;
    }
    timeAfter = time_get_ms();
    set_up_key_time=0.0;
    PRINTF("Time  was  taken  in  decrypt  %f  ms.\r\n\r\n",timeAfter  -
timeBefore);

    PRINTF("AES-128 CBC decryption finished. Speed %f MB/s.\r\n",
           mmcau_get_throughput(timeAfter        -        timeBefore,
CYCLES_FOR_THROUGHPUT * length));
```

```
    throughtput=mmcau_get_throughput(timeAfter      -      timeBefore,
CYCLES_FOR_THROUGHPUT * length);

    printf("AES-128 CBC encryption cycles per byte %.3f cycles per
byte.\r\n\r\n",
           get_cycles_per_byte(throughtput));;


    /* Print decrypted string */
    PRINTF("Decrypted string :\r\n");
    mmcau_print_msg(g_result, length);
    PRINTF("\r\n");


    PRINTF("---------------------------------- AES-192-CBC method --
-----------------------------------\r\n");

    /*   ENCRYPTION   */
    PRINTF("AES-192 CBC Encryption of %d bytes.\r\n", length);

    /* Call AES_cbc encryption */
    cycles     = CYCLES_FOR_THROUGHPUT;

    timeBefore1 = time_get_ms();
    while (cycles)
    {
      //timeBefore1 = time_get_ms();
      status = mmcau_encrypt_aes_cbc(g_aesKey192, AES192, g_testFull,
g_output, length, g_aesIV);
        //timeAfter1 = time_get_ms();
        //elapsed_time=timeAfter1 - timeBefore1;
        //total_time+=elapsed_time;
        if (status != MMCAU_OK)
        {
                PRINTF("AES-192 CBC encryption failed !\r\n");
                return;

        }
        cycles--;
    }
    timeAfter1 = time_get_ms();
    elapsed_time=timeAfter1 - timeBefore1;

    PRINTF("ms             per             operation:             %f
ms.\r\n\r\n",(elapsed_time/CYCLES_FOR_THROUGHPUT));
    PRINTF("-----192:   set   up   key   time   total   encrypt   %f
ms.\r\n\r\n",set_up_key_time/CYCLES_FOR_THROUGHPUT);
    set_up_key_time=0.0;

        /* Result message */


    cycles     = CYCLES_FOR_THROUGHPUT;
    timeBefore = time_get_ms();
    while (cycles)
    {
        status    =    mmcau_encrypt_aes_cbc(g_aesKey192,    AES192,
g_testFull, g_output, length, g_aesIV);
        if (status != MMCAU_OK)
        {
            PRINTF("AES-192 CBC encryption failed !\r\n");
```

```
            return;
        }
        cycles--;
    }
    timeAfter = time_get_ms();
    PRINTF("Time was taken in encrypt %f ms.\r\n\r\n",timeAfter -
timeBefore);
    set_up_key_time=0.0;

    /* Result message */
    PRINTF("AES-192 CBC encryption finished. Speed %f MB/s.\r\n",
           mmcau_get_throughput(timeAfter        -        timeBefore,
CYCLES_FOR_THROUGHPUT * length));

    throughtput=mmcau_get_throughput(timeAfter      -      timeBefore,
CYCLES_FOR_THROUGHPUT * length);

    printf("AES-192 CBC encryption cycles per byte %.3f cycles per
byte.\r\n\r\n",
           get_cycles_per_byte(throughtput));



    /*   DECRYPTION   */
    PRINTF("AES-192 CBC Decryption of %d bytes.\r\n", length);


    cycles     = CYCLES_FOR_THROUGHPUT;

    while (cycles)
    {
      timeBefore1 = time_get_ms();
        status = mmcau_decrypt_aes_cbc(g_aesKey192, AES192, g_output,
g_result, length, g_aesIV);
        timeAfter1 = time_get_ms();
        elapsed_time=timeAfter1 - timeBefore1;
        total_time+=elapsed_time;
        if (status != MMCAU_OK)
        {
            PRINTF("AES-192 CBC decryption failed !\r\n");
            return;
        }
        cycles--;
    }

    PRINTF("ms              per              operation:              %f
ms.\r\n\r\n",(total_time/CYCLES_FOR_THROUGHPUT));
    PRINTF("-----192:   set   up   key   time   total   decrypt   %f
ms.\r\n\r\n",set_up_key_time/CYCLES_FOR_THROUGHPUT);
    set_up_key_time=0.0;

    /* Call AES_cbc decryption */
    cycles     = CYCLES_FOR_THROUGHPUT;
    timeBefore = time_get_ms();
    while (cycles)
    {
        status = mmcau_decrypt_aes_cbc(g_aesKey192, AES192, g_output,
g_result, length, g_aesIV);
        if (status != MMCAU_OK)
        {
```

```
            PRINTF("AES-192 CBC decryption failed !\r\n");
            return;
        }
        cycles--;
    }
    timeAfter = time_get_ms();
    PRINTF("Time was taken in decrypt %f ms.\r\n\r\n",timeAfter -
timeBefore);
    set_up_key_time=0.0;

    /* Result message */
    PRINTF("AES-192 CBC decryption finished. Speed %f MB/s.\r\n",
          mmcau_get_throughput(timeAfter       -       timeBefore,
CYCLES_FOR_THROUGHPUT * length));

    throughtput=mmcau_get_throughput(timeAfter     -     timeBefore,
CYCLES_FOR_THROUGHPUT * length);

    printf("AES-192 CBC encryption cycles per byte %.3f cycles per
byte.\r\n\r\n",
          get_cycles_per_byte(throughtput));

    /* Print decrypted string */
    PRINTF("Decrypted string :\r\n");
    mmcau_print_msg(g_result, length);
    PRINTF("\r\n");


    PRINTF("---------------------------------- AES-256-CBC method --
-----------------------------------\r\n");

    /*   ENCRYPTION   */
    PRINTF("AES-256 CBC Encryption of %d bytes.\r\n", length);

    /* Call AES_cbc encryption */
    cycles     = CYCLES_FOR_THROUGHPUT;
    timeBefore1 = time_get_ms();
    while (cycles)
    {
      //timeBefore1 = time_get_ms();
        status    =    mmcau_encrypt_aes_cbc(g_aesKey256,    AES256,
g_testFull, g_output, length, g_aesIV);
        //timeAfter1 = time_get_ms();
        //elapsed_time=timeAfter1 - timeBefore1;
        //total_time+=elapsed_time;
        if (status != MMCAU_OK)
        {
             PRINTF("AES-256 CBC encryption failed !\r\n");
             return;
        }
        cycles--;
    }
    timeAfter1 = time_get_ms();
    elapsed_time=timeAfter1 - timeBefore1;
    PRINTF("ms             per             operation:             %f
ms.\r\n\r\n",(elapsed_time/CYCLES_FOR_THROUGHPUT));
    PRINTF("-----256set    up    key    time    total    encrypt    %f
ms.\r\n\r\n",set_up_key_time/CYCLES_FOR_THROUGHPUT);
    set_up_key_time=0.0;

    /* Call AES_cbc encryption */
```

```
    cycles     = CYCLES_FOR_THROUGHPUT;
    timeBefore = time_get_ms();
    while (cycles)
    {
        status    =    mmcau_encrypt_aes_cbc(g_aesKey256,    AES256,
g_testFull, g_output, length, g_aesIV);
        if (status != MMCAU_OK)
        {
            PRINTF("AES-256 CBC encryption failed !\r\n");
            return;
        }
        cycles--;
    }
    timeAfter = time_get_ms();
    PRINTF("Time  was  taken  in  encrypt  %f  ms.\r\n\r\n",timeAfter  -
timeBefore);
    set_up_key_time=0.0;


    /* Result message */
    PRINTF("AES-256 CBC encryption finished. Speed %f MB/s.\r\n\r\n",
           mmcau_get_throughput(timeAfter        -        timeBefore,
CYCLES_FOR_THROUGHPUT * length));


    throughtput=mmcau_get_throughput(timeAfter     -     timeBefore,
CYCLES_FOR_THROUGHPUT * length);

    printf("AES-256 CBC encryption cycles per byte %.3f cycles per
byte.\r\n\r\n",
           get_cycles_per_byte(throughtput));

    /*   DECRYPTION   */
    PRINTF("AES-256 CBC Decryption of %d bytes.\r\n", length);

    cycles     = CYCLES_FOR_THROUGHPUT;
    timeBefore1 = time_get_ms();
    while (cycles)
    {
      //timeBefore1 = time_get_ms();
        status = mmcau_decrypt_aes_cbc(g_aesKey256, AES256, g_output,
g_result, length, g_aesIV);
        //timeAfter1 = time_get_ms();
        //elapsed_time=timeAfter1 - timeBefore1;
        //total_time+=elapsed_time;
        if (status != MMCAU_OK)
        {
            PRINTF("AES-256 CBC decryption failed !\r\n");
            return;
        }
        cycles--;
    }
    timeAfter1 = time_get_ms();
    elapsed_time=timeAfter1 - timeBefore1;
    PRINTF("ms             per             operation:             %f
ms.\r\n\r\n",(total_time/CYCLES_FOR_THROUGHPUT));
    PRINTF("-----256:   set   up   key   time   total   decrypt   %f
ms.\r\n\r\n",set_up_key_time/CYCLES_FOR_THROUGHPUT);
    set_up_key_time=0.0;
```

```
    /* Call AES_cbc decryption */
    cycles     = CYCLES_FOR_THROUGHPUT;
    timeBefore = time_get_ms();
    while (cycles)
    {
        status = mmcau_decrypt_aes_cbc(g_aesKey256, AES256, g_output,
g_result, length, g_aesIV);
        if (status != MMCAU_OK)
        {
            PRINTF("AES-256 CBC decryption failed !\r\n");
            return;
        }
        cycles--;
    }
    timeAfter = time_get_ms();
    PRINTF("Time  was  taken  in  decrypt  %f  ms.\r\n\r\n",timeAfter  -
timeBefore);
    set_up_key_time=0.0;

    /* Result message */
    PRINTF("AES-256 CBC decryption finished. Speed %f MB/s.\r\n",
           mmcau_get_throughput(timeAfter        -        timeBefore,
CYCLES_FOR_THROUGHPUT * length));


    throughtput=mmcau_get_throughput(timeAfter     -     timeBefore,
CYCLES_FOR_THROUGHPUT * length);

    printf("AES-256  CBC  encryption  cycles  per  byte  %.3f  cycles  per
byte.\r\n\r\n",
           get_cycles_per_byte(throughtput));

    /* Print decrypted string */
    PRINTF("Decrypted string :\r\n");
    mmcau_print_msg(g_result, length);
    PRINTF("\r\n");


    /* Format console output */
    PRINTF("\r\n\r\n");

    /* Goodbye message */
    PRINTF("..............  THE  END  OF  THE  MMCAU  DRIVER  EXAMPLE
................................\r\n");
}

/*!
 * @brief MMCAU example.
 */
int main(void)
{
    /* Init hardware*/
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();

    /* Init time measurement. SysTick method deployed. */
    if (time_config(false))
    {
        PRINTF("ERROR in SysTick Configuration\r\n");
    }
```

```
    else
    {
        /* Call example task */
        mmcau_example_task();
    }

    while (1)
    {
    }
}
```