Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή
Στην Ασφάλεια Υπολογιστών και Δικτύων

Μελέτη "Lightweight" Κρυπτογραφικών Αλγορίθμων Ως Προς Την Απόδοσή Τους

Παναγιώτης Ποδηματάς

Επιβλέπων Καθηγητής
Κωνσταντίνος Λιμνιώτης

Μάιος 2020

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μελέτη "Lightweight" Κρυπτογραφικών Αλγορίθμων Ως Προς Την Απόδοσή Τους

Παναγιώτης Ποδηματάς

Επιβλέπων Καθηγητής
Κωνσταντίνος Λιμνιώτης

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στην Ασφάλεια Υπολογιστών και Δικτύων

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Μάιος 2020

Περίληψη

Οι λεγόμενοι «lightweight» κρυπτογραφικοί αλγόριθμοι αποτελούν μία σημαντική κατηγορία κρυπτογραφικών αλγορίθμων που αποκτά ιδιαίτερη βαρύτητα ιδίως στα τελευταία χρόνια, με την εξέλιξη του χώρου του Διαδικτύου των Πραγμάτων (Internet-of-Things). Ήδη άλλωστε ο οργανισμός NIST έχει εκκινήσει διαδικασία καθορισμού πρότυπου αλγορίθμου σε αυτήν την κατηγορία.

Η παρούσα διατριβή εστιάζει στη μελέτη των lightweight αλγορίθμων ιδίως ως προς την απόδοσή τους και την αξιολόγηση αυτής σε σημερινά πραγματικά περιβάλλοντα. Συγκεκριμένα, μελετώνται πρόσφατα ανεπτυγμένοι αλγόριθμοι αυτής της κατηγορίας οι οποίοι θεωρούνται υποσχόμενοι και με ελεύθερα διαθέσιμες υλοποιήσεις (open source), όπως – ως μελέτη περίπτωσης – η οικογένεια αλγορίθμων SATURNIN, η οποία έχει υποβληθεί στον ως άνω διαγωνισμό του NIST, ως προς το βαθμό στον οποίο μπορούν να ενσωματωθούν σε σύγχρονα δικτυακά πρωτόκολλα ασφαλείας (όπως στο TLS ή/και σε εφαρμογές VoIP), σε διάφορα περιβάλλοντα υλοποίησης, με βάση το όφελος ή όχι στην απόδοση του πρωτοκόλλου η οποία θα παρατηρηθεί. Παράλληλα γίνεται και περιγραφή του επιπέδου ασφαλείας που επιτυγχάνεται με τη χρήση των αλγορίθμων αυτών στα συγκεκριμένα πρωτόκολλα, λαμβάνοντας υπόψη και την ανθεκτικότητά τους στη μετα-κβαντική εποχή. .Για την αξιολόγηση της απόδοσης του SATURNIN, υλοποιήθηκε κατάλληλο πειραματικό περιβάλλον προκειμένου να μετρηθεί η απόδοση του SATURNIN σε σύγκριση με τον αντίστοιχο πρότυπο αλγόριθμο AES: το περιβάλλον αυτό είναι περιβάλλον με περιορισμούς, έτσι ώστε να αντιστοιχεί σε εφαρμογές στις οποίες απευθύνεται ιδίως η lightweight κρυπτογράφηση. Τα πειράματα κατέδειξαν ότι ο SATURNIN – του οποίου η σχεδίαση βασίστηκε στη σχεδίαση του AES - μπορεί να είναι μέχρι και περίπου δύο φορές πιο γρήγορος από τον AES σε τέτοια περιβάλλοντα με περιορισμούς.

Summary

«Lightweight» cryptography is an important field in modern cryptography nowadays, mainly because of the need to produce new algorithms for a whole new category of devices, the so called 'Internet-of-Things'. NIST has already initiated a process to define a new suite of standards of «Lightweight» cryptographic algorithms.

In this thesis, we will study the «Lightweight» algorithms and we will focus on their performance in today's computer environments. More precisely, focusing – as a case study – on a family of lightweight ciphers being called "SATURNIN", which has been submitted in the aforementioned NIST's competition and it is available in open source, we analyze their ability to be used in modern security protocols like TLS, towards to assess their overall performance.. Moreover, the security properties of SATURNIN are also discussed, taking also into account its security in the post-quantum era. To evaluate the performance of SATURNIN, an appropriate experimental environment has been developed, towards measuring the speed of SATURNIN in comparison with the speed of the current standard AES: this environment is restrictioned, since lightweight cryptography focuses exactly on such types of environments. Our experimental analysis illustrates that SATURNIN – whose design is inspired by the design of AES – can be almost two times faster than AES in such environments.

Ευχαριστίες

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα την μεταπτυχιακή μου διατριβή, καθηγητή Κωνσταντίνο Λιμνιώτη. Ο τρόπος διδασκαλίας του και η εξαιρετική μεταδοτικότητά του κατά τη διάρκεια του μαθήματος της κρυπτογραφίας με έκαναν να αγαπήσω θερμά ένα τομέα με τον οποίο δεν είχα ασχοληθεί ποτέ στο παρελθόν. Και φυσικά για την καθοδήγηση αλλά και την επιστημονική του συμβολή κατά τη διάρκεια εκπόνησης της διατριβής.

Ευχαριστώ πολύ την οικογένειά μου. Ξέρουν γιατί….

Πίνακας περιεχομένων

Κεφάλαιο 1
Εισαγωγή

Οι κρυπτογραφικοί αλγόριθμοι που χρειάζονται για να διασφαλίσουν την ασφαλή επικοινωνία στο Διαδίκτυο έχουν, γενικά, μεγάλες απαιτήσεις σε υπολογιστική ισχύ. Οι σύγχρονοι υπολογιστές είναι αρκετά ισχυροί για να ανταπεξέλθουν στον έξτρα φόρτο. Ακόμα και τα smartphones ή τα tablets είναι πια τόσο ισχυρά που δεν έχουν πρόβλημα να καλύψουν αυτές τις απαιτήσεις.

Υπάρχει όμως ένας ολόκληρος «κόσμος» από μικροσυσκευές που κατακλύζουν το Διαδίκτυο κατά εκατοντάδες χιλιάδες κάθε χρόνο και οι οποίες δεν έχουν μεγάλη υπολογιστική ισχύ. Μιλάμε για το «Διαδίκτυο των πραγμάτων» ή Internet of Things (IoT), που αποτελείται από μικρά υπολογιστικά boards, φτιαγμένα κυρίως να εκτελούν μία συγκεκριμένη εργασία, με εφαρμογές όμως, που αποστέλλουν δεδομένα μέσω Διαδικτύου σε άλλα, μεγαλύτερα υπολογιστικά συστήματα. Η εφαρμογή λοιπόν των υπαρχόντων κρυπτογραφικών αλγόριθμων σε αυτές τις μικροσυσκευές απαιτεί πολλούς πόρους (απαιτήσεις λειτουργίας σε χαμηλή επεξεργαστική ισχύ, χαμηλή μνήμη ή\και πολύ μικρή επιφάνεια υλοποίησης) με αποτέλεσμα να μην είναι πάντα ευχερής η υλοποίηση ευρέως γνωστών, και ασφαλών, κρυπτογραφικών συστημάτων σε τέτοια περιβάλλοντα με περιορισμούς.

Μια απλή λύση θα ήταν να μειώσουμε τα επίπεδα ασφάλειας των υπαρχόντων αλγορίθμων μειώνοντας την πολυπλοκότητά τους και κάνοντας τις συναρτήσεις τους πιο απλές. Π.χ. αν μειώναμε τον αριθμό των «γύρων» που χρησιμοποιεί ο AES (θα μιλήσουμε για αυτόν παρακάτω), ή κάνοντας πιο απλή τη συνάρτηση που διαχειρίζεται τους «γύρους», θα μπορούσαμε να μειώσουμε την συνολική απαίτηση για επεξεργαστική ισχύ.

Η διάδοσή όμως των IoT είναι τόσο μεγάλη, που πια χρησιμοποιούνται και για εφαρμογές με τις οποίες μεταφέρονται ευαίσθητα δεδομένα μέσω Διαδικτύου. Η εποχή που τα IoT μας έστελναν στο κινητό μόνο δεδομένα όπως η θερμοκρασία και η υγρασία, έχει περάσει και, πλέον, χρησιμοποιούνται ευρέως σε ευαίσθητες εφαρμογές, όπως στη φύλαξη κτιρίων, σε έλεγχο κυκλοφορίας, στην ιατρική, με αποτέλεσμα να μεταδίδονται δεδομένα που χρήζουν μέγιστης ασφάλειας. Άρα οι μείωση των επιπέδων ασφαλείας στους υπάρχοντες αλγόριθμους είναι μια λύση που δεν μπορεί να γίνει αποδεκτή.

Ταυτόχρονα, η νέα εποχή στην επιστήμη των υπολογιστών ονομάζεται «κβαντικός υπολογιστής». Η νέα γενιά αυτή των υπολογιστών, με τη διαφορετική αρχή λειτουργίας, δεν θα αντικαταστήσει ακόμα τους κλασσικούς υπολογιστές σε όλο το φάσμα των εφαρμογών τους, αλλά υπόσχεται δραματική αύξηση της υπολογιστικής ισχύς σε ένα τομέα που μας ενδιαφέρει ιδιαίτερα. Στην «κρυπτανάλυση»! Έτσι, οι κβαντικοί υπολογιστές θα μπορούν να «σπάνε» με ευκολία σχεδόν όλους τους αλγόριθμους δημοσίου κλειδιού που γνωρίζουμε σήμερα και θα επιταχύνουν στο διπλάσιο την επεξεργασία για τους συμμετρικούς αλγόριθμους.

Όλα τα παραπάνω δημιούργησαν την ανάγκη για την κατασκευή μιας νέας γενιάς αλγορίθμων οι οποίοι να είναι πολύ γρήγοροι, με μικρές ανάγκες υπολογιστικής ισχύος, αλλά ταυτόχρονα να ξεπερνούν σε επίπεδα ασφάλειας τους υπάρχοντες αλγόριθμους και να μπορούν να διατηρούν τα επίπεδα ασφάλειάς τους και στη μετακβαντική εποχή.

Οι λεγόμενοι **«lightweight» κρυπτογραφικοί αλγόριθμοι** λοιπόν, σκοπό έχουν να πληρούν τις παραπάνω προϋποθέσεις και αποτελούν μία σημαντική κατηγορία κρυπτογραφικών αλγορίθμων που αποκτά ιδιαίτερη βαρύτητα ιδίως στα τελευταία χρόνια, με την εξέλιξη του χώρου του Διαδικτύου των Πραγμάτων (Internet-of-Things). Ήδη άλλωστε ο οργανισμός NIST (National Institute of Standards and Technology) έχει εκκινήσει διαδικασία καθορισμού πρότυπου αλγορίθμου σε αυτήν την κατηγορία

1.1 Ερευνητικά ερωτήματα

Η διατριβή εστιάζει στους "lightweight" αλγορίθμους, δίνοντας έμφαση στην τρέχουσα ερευνητική δραστηριότητα που σχετίζεται με τον υπό εξέλιξη διαγωνισμό του NIST για προσδιορισμό προτύπων αλγορίθμων. Ειδικότερα, θα προσπαθήσει να απαντήσει στο κατά πόσο συγκεκριμένοι lightweight κρυπτογραφικοί αλγόριθμοι, και ειδικότερα – ως μελέτη περίπτωσης - η σουίτα αλγορίθμων «Saturnin», μία εκ των υποψηφίων προς προτυποποίηση η οποία έχει υποβληθεί στον NIST, είναι αρκετά ισχυρή και μπορεί να ανταπεξέλθει στις απαιτήσεις της «lightweight» κρυπτογραφίας. Πιο συγκεκριμένα, πέραν του ότι μελετώνται ιδιότητες ασφαλείας της ως προς το αν είναι οι αλγόριθμοι αυτοί είναι ικανοί να διατηρήσουν ή και να αυξήσουν τα επίπεδα ασφάλειας των υπαρχόντων αλγορίθμων, πραγματοποιείται η αξιολόγηση της απόδοσής τους, σε ταχύτητα αλλά και σε κατανάλωση πόρων, σε περιβάλλοντα περιορισμένης επεξεργαστικής ισχύος (π.χ., IoT). Τέλος, εξετάζεται και υπό ποιες προϋποθέσεις είναι αρκετά ισχυροί για να μπορούν να θεωρηθούν ανθεκτικοί σε επιθέσεις από κβαντικούς υπολογιστές. Ο λόγος για τον οποίο μελετάται η συγκεκριμένη σουίτα ως μελέτη περίπτωσης είναι ότι έχει πολλά κοινά χαρακτηριστικά στη σχεδίασή της, με το σημερινό καθολικό πρότυπο κρυπτογράφησης (AES).

1.2 Μεθοδολογία της έρευνας

Η παρούσα διατριβή θα εστιάσει στη μελέτη των lightweight αλγορίθμων ιδίως ως προς την απόδοσή τους και την αξιολόγηση αυτής σε σημερινά πραγματικά περιβάλλοντα. Θα επιχειρηθεί να γίνει μέτρηση και αξιολόγηση της αποδοτικότητας των αλγορίθμων «Saturnin» σε σχέση με υπάρχοντες αλγορίθμους που θεωρούνται ανθεκτικοί σε περιβάλλον κβαντικών υπολογιστών, όπως π.χ. ο AES 256bit. Για τον σκοπό αυτό θα χρησιμοποιηθεί board με περιορισμένη υπολογιστική ισχύ και θα αναπτυχθεί αντίστοιχο λογισμικό.

Συγκεκριμένα, θα υλοποιήσουμε τον AES και τον Saturnin στο ESP8266 board που είναι μια μικροσυσκευή με περιορισμένη επεξεργαστική ισχύ (max 160MHz ρολόι για την mcu) και ελάχιστη μνήμη RAM (μόλις 80Kbytes). Η μικροσυσκευή έχει τη δυνατότητα να συνδεθεί με το Διαδίκτυο με το Wi-Fi transceiver που διαθέτει και έτσι θα επικοινωνεί με μία εφαρμογή desktop υλοποιημένη σε QT framework(γλώσσα προγραμματισμού C++) και θα ανταλλάσσουν κρυπτομηνύματα χρησιμοποιώντας και τους δύο αλγορίθμους, δηλαδή τον AES και τον Saturnin. Έτσι, καταγράφοντας τους χρόνους που χρειάζεται ο κάθε αλγόριθμος, θα καταλήξουμε σε συμπεράσματα που αφορούν την απόδοσή τους.

1.3 Δομή διατριβής

Η παρούσα εργασία ξεκινά εισαγωγικά με την περιγραφή των προβλημάτων που αντιμετωπίζουν οι σημερινοί κρυπτογραφικοί αλγόριθμοι όταν εφαρμόζονται σε υπολογιστές με μικρή υπολογιστική ισχύ όπως είναι τα Internet-of-Things. Επιπρόσθετα παρουσιάζεται και η μελλοντική απαίτηση για αλγόριθμους που παρουσιάζουν αντοχή στη μετακβαντική κρυπτανάλυση και πως τα παραπάνω μπορούν να λυθούν με την εφαρμογή των λεγόμενων «lightweight» κρυπτογραφικών αλγόριθμων.

Στο κεφάλαιο 2 κάνουμε μια σύντομη εισαγωγή στην κρυπτογραφία και εστιάζουμε στις δύο βασικές κατηγορίες της μοντέρνας κρυπτογραφίας. Στην «συμμετρική» και στην «ασύμμετρη». Όλοι οι αλγόριθμοι που χρησιμοποιούνται σήμερα ανήκουν σε μία από τις δύο κατηγορίες και ειδικά για την επικοινωνία στο Διαδίκτυο, το πρότυπο TLS που είναι ο de facto τρόπος για τη διατήρηση της ασφάλειας, χρησιμοποιεί οικογένειες αλγορίθμων από «συμμετρική» και «ασύμμετρη» κρυπτογραφία. Τέλος, αναλύουμε πιο λεπτομερειακά τον αλγόριθμο AES γιατί είναι ο αλγόριθμος αναφοράς του «lightweight» αλγορίθμου Saturnin που θέλουμε να μετρήσουμε ως προς την απόδοση.

Στο κεφάλαιο 3 παρουσιάζουμε το Διαδίκτυο των πραγμάτων (Internet of Things). Συγκεκριμένα, περιγράφονται οι διαφορές των συσκευών αυτών από ανάλογες συσκευές του παρελθόντος, και η

δυνατότητα χρήσης του Διαδικτύου από μέρους τους, για την ανταλλαγή δεδομένων μεταξύ τους αλλά και την αποστολή δεδομένων σε μεγαλύτερους υπολογιστές για μαζική επεξεργασία. Αναφέρουμε στη συνέχεια τα μεγαλύτερα προβλήματα που αντιμετωπίζουν οι συσκευές του Internet of Things κατά την επικοινωνία τους μέσω Διαδικτύου, όσον αφορά την ασφάλεια και την ακεραιότητα των δεδομένων. Στη συνέχεια αναλύουμε τα πρωτόκολλα SSL/TLS με ιδιαίτερη έμφαση στο TLS καθώς το τελευταίο είναι ο πλέον βασικός τρόπος ασφαλούς επικοινωνίας στο Διαδίκτυο σήμερα. Τέλος παρουσιάζουμε τις μεθόδους που θα μπορούσε να χρησιμοποιήσει ένας κρυπταναλυτής για να παραβιάσει την ασφάλεια των κρυπτογραφικών συστημάτων που θα μπορούσαν να χρησιμοποιηθούν στα Internet of Things.

Στο 4ο κεφάλαιο παρουσιάζουμε τους κβαντικούς υπολογιστές και την αρχή λειτουργίας τους. Εξηγούμε πως με τη βοήθεια τους, η κρυπτανάλυση θα αυξήσει δραματικά την υπολογιστική ισχύ που διαθέτει, με αποτέλεσμα όλοι οι αλγόριθμοι δημοσίου κλειδιού να μπορούν να «σπάσουν» με μεγάλη ευκολία και οι αλγόριθμοι συμμετρικού κλειδιού να θεωρούνται ως αλγόριθμοι με υποδιπλάσιο επίπεδο ασφάλειας σε σχέση με αυτό που έχουν τώρα.

Ακολούθως, στο κεφάλαιο 5 παρουσιάζεται η "Lightweight" Κρυπτογραφία. Αναφέρουμε τις διαφορές της από τη υπάρχουσα «συμβατική» κρυπτογραφία, τα χαρακτηριστικά που πρέπει να έχουν οι αλγόριθμοι και ποιες παραδοχές πρέπει να γίνουν κατά την υλοποίηση τους με απώτερο σκοπό την αύξηση της απόδοσης. Καταγράφουμε τις κατηγορίες των "Lightweight" αλγορίθμων και αναλύουμε τις διάφορες προσεγγίσεις που υιοθετήθηκαν για κάθε κατηγορία. Τέλος, αναφέρουμε τις παραδείγματα διαθέσιμων υλοποιήσεων για κάθε κατηγορία.

Στο κεφάλαιο 6 αναλύουμε τον Saturnin, μια οικογένεια αλγορίθμων που πληροί τις προϋποθέσεις για να ανήκει στην κατηγορία της "Lightweight" Κρυπτογραφίας. Παρουσιάζουμε τα χαρακτηριστικά του, τον τρόπο λειτουργίας του, τη σχέση του με τον AES και πως τον βελτιώνει σημαντικά σε θέματα ασφάλειας, Επίσης κάνουμε εκτενή αναφορά για την ασφάλεια του αλγορίθμου ενάντια σε επιθέσεις, τόσο μέσω «κλασσικής» κρυπτανάλυσης, όσο και μέσω «μετακβαντικής».

Στο κεφάλαιο 7 παρουσιάζεται η δοκιμή απόδοσης του Saturnin σε σχέση με τον AES σε περιβάλλον με Internet of Things. Περιγράφουμε το περιβάλλον προγραμματισμού και τα αντίστοιχα εργαλεία λογισμικού που χρησιμοποιήθηκαν, δίνουμε λεπτομέρειες για τον τρόπο λειτουργίας των εφαρμογών και παρουσιάζουμε τα αποτελέσματα, κάνοντας σύγκριση των χρόνων που χρειάστηκαν οι δύο αλγόριθμοι για κρυπτογράφηση και αποστολή των κρυπτομηνυμάτων.

Τέλος, η σύνοψη της εργασίας που έγινε, στα πλαίσια της παρούσας διατριβής, καθώς και τα συμπεράσματα και οι σκέψεις για μελλοντική έρευνα, αναφέρονται στο κεφάλαιο 8.

Κεφάλαιο 2
Κρυπτογραφία

«Ουδέν κρυπτόν υπό τον ήλιον»…….Όχι πια…!

Κρυπτογραφία είναι ο κλάδος όπου ασχολείται με μαθηματικούς μετασχηματισμούς προκειμένου να εξασφαλιστεί η ασφάλεια της πληροφορίας. Είναι δηλαδή, η μελέτη τεχνικών που βασίζονται σε μαθηματικά προβλήματα δύσκολο να λυθούν, με σκοπό την εξασφάλιση της ασφάλειας (εμπιστευτικότητα, ακεραιότητα, αυθεντικότητα) των δεδομένων.

Κρυπτανάλυση είναι η μελέτη μαθηματικών τεχνικών για την προσβολή κρυπτογραφικών τεχνικών ή υπηρεσιών ασφάλειας.

Κρυπτολογία είναι ο συνδυασμός της κρυπτογραφίας και κρυπτανάλυσης σε ένα ενιαίο επιστημονικό κλάδο.

Εφαρμογή της κρυπτογραφίας είναι η **κρυπτογράφηση**. Κρυπτογράφηση είναι ο μετασχηματισμός δεδομένων σε μορφή που να είναι αδύνατον να διαβαστεί χωρίς τη γνώση της σωστής ακολουθίας bit. Η ακολουθία bit καλείται "**κλειδί**" και χρησιμοποιείται σε συνδυασμό με κατάλληλο αλγόριθμο / συνάρτηση. Η αντίστροφη διαδικασία είναι η **αποκρυπτογράφηση** και απαιτεί γνώση του κλειδιού. Σκοπός της κρυπτογράφησης είναι να εξασφαλίσει το απόρρητο των δεδομένων κρατώντας τα κρυφά από όλους όσους έχουν πρόσβαση σε αυτά.

Η πρώτη περίοδος κρυπτογραφίας υπολογίζεται μεταξύ 1900 π.Χ. – 1900 μ.Χ., κατά τη διάρκεια της οποίας αναπτύχθηκε ένας μεγάλος αριθμός μεθόδων και αλγορίθμων κρυπτογράφησης βασιζόμενοι κυρίως σε απλές αντικαταστάσεις γραμμάτων. Στις μέρες μας όμως, όλα αυτά τα συστήματα έχουν κρυπταναλυθεί και έχει αποδειχθεί ότι, εφόσον ένα μεγάλο κομμάτι του κρυπτογραφικού μηνύματος είναι γνωστό, τότε η επανάκτηση του αρχικού μηνύματος είναι σχετικά εύκολη.

Η κρυπτογραφία χρησιμοποιήθηκε πρώτη φορά για στρατιωτικούς λόγους από τους Σπαρτιάτες. Τον 5ο αιώνα π.Χ. ανακάλυψαν τη πρώτη κρυπτογραφική συσκευή, τη «σκυτάλη», στην οποία χρησιμοποίησαν την μέθοδο της μετάθεσης. Σύμφωνα με τον Πλούταρχο, η «Σπαρτιατική Σκυτάλη» ήταν μια ξύλινη ράβδος ορισμένης διαμέτρου, στην οποία βρισκόταν τυλιγμένη μια λωρίδα

περγαμηνής. Το κείμενο ήταν γραμμένο σε στήλες, με ένα γράμμα σε κάθε έλικα, έτσι όταν ξετύλιγαν τη λωρίδα, το κείμενο ήταν δυσνόητο εξαιτίας της αναδιάταξης των γραμμάτων. Το κλειδί για την αποκρυπτογράφηση ήταν η διάμετρος της σκυτάλης

Εικόνα 1. Σπαρτιατική Σκυτάλη

Αν και δεν γνωρίζουμε πότε χρησιμοποιήθηκαν αρχικά τα συστήματα γραπτής αντικατάστασης γραμμάτων, τα συναντάμε στην Ρωμαϊκή Αυτοκρατορία, κυρίως επί βασιλεία του Ιούλιου Καίσαρα, ο οποίος αλληλογραφούσε με τον Κικέρωνα και άλλους φίλους του, αντικαθιστώντας τα γράμματα του κειμένου, με γράμματα μετατοπισμένα κατά 3 θέσεις μετά, στο Λατινικό Αλφάβητο. Στις μέρες μας, τα συστήματα που στηρίζονται στην αντικατάσταση γραμμάτων με άλλα γράμματα που είναι μετατοπισμένα κατά ένα καθορισμένο αριθμό θέσεων δεξιά ή αριστερά της αλφαβήτου, ονομάζονται κρυπτοσυστήματα αντικατάστασης του Καίσαρα.

Η δεύτερη περίοδος της κρυπτογραφίας χρονολογείται στις αρχές του 20ού αιώνα μέχρι το 1950. Καλύπτει και τους δύο παγκοσμίους πολέμους οι οποίοι αποτέλεσαν και την αιτία, λόγω της ανάγκης για ασφαλή μετάδοση σημαντικών πληροφοριών, για την ραγδαία ανάπτυξη της. Εκείνη την περίοδο τα κρυπτοσυστήματα αρχίζουν να γίνονται πιο περίπλοκα και να αποτελούνται από μηχανικές και ηλεκτρομηχανικές κατασκευές, τις «κρυπτομηχανές».

Παρά το γεγονός ότι τα συστήματα εκείνης της περιόδου ήταν αρκετά πολύπλοκα, η κρυπτανάλυση τους ήταν συνήθως επιτυχημένη. Οι Γερμανοί π.χ. χρησιμοποιούσαν κυρίως ένα σύστημα που είναι γνωστό ως Enigma την οποία κατάφερε να παραβιάσει ο Marian Rejewski το 1932 στην Πολωνία, χρησιμοποιώντας θεωρητικά μαθηματικά.

Εικόνα 2. Η μηχανή Enigma

Το 1939 ο γερμανικός στρατός έκανε κάποιες αλλαγές, με αποτέλεσμα οι Πολωνοί να μην είναι σε θέση να παρακολουθήσουν, καθώς η αποκρυπτογράφηση πλέον απαιτούσε περισσότερους πόρους από αυτούς που είχαν διαθέσιμους. Η γνώση τους μεταβιβάστηκε στους Γάλλους και τους Βρετανούς και ο Alan Turing οδήγησε σε ένα μεγάλο αριθμό αποκρυπτογραφήσεων διαφόρων παραλλαγών του Enigma, με την βοήθεια ενός υπολογιστή βρετανικής κατασκευής, του Colossus, ο οποίος καταστράφηκε μετά το τέλος του πολέμου.

Η τρίτη περίοδος χρονολογείται από το 1950 μέχρι σήμερα και χαρακτηρίζεται από την ραγδαία ανάπτυξη στους κλάδους των μαθηματικών, της μικροηλεκτρονικής και των υπολογιστικών συστημάτων. Η περίοδος της σύγχρονης κρυπτογραφίας ξεκινά με τον πατέρα των μαθηματικών συστημάτων κρυπτογραφίας, τον Claude Shannon, ο οποίος δημοσίευσε το άρθρο «Communication Theory of Secrecy Systems» στο τεχνικό περιοδικό Bell System [1] και λίγο αργότερα το άρθρο «A Mathematical Theory of Communication» [2].

Στα μέσα της δεκαετίας του 1970, πραγματοποιήθηκαν δύο σπουδαίες δημοσιεύσεις. Η πρώτη ήταν αυτή του σχεδίου προτύπου κρυπτογράφησης DES (Data Encryption Standard) [3] και η δεύτερη η θεωρία για την κρυπτογράφηση δημοσίου κλειδιού (Public key encryption) των Diffie-Hellman [4]. Ο DES χρησιμοποιήθηκε προκειμένου να αναπτυχθούν ασφαλείς ηλεκτρονικές εγκαταστάσεις επικοινωνίας σε μεγάλες επιχειρήσεις. Ο DES αποτέλεσε τον πρώτο προσιτό αλγόριθμο κρυπτογράφησης που εγκρίθηκε από μια εθνική αντιπροσωπεία όπως η NSA

Μετά από αναγγελία του NIST (National Institute of Standards and Technology) [5] ο DES αντικαταστάθηκε από τον AES το 2001. Ο DES, λόγω του μικρού μεγέθους κλειδιού (56 bit), έπαψε να θεωρείται ασφαλής (κατάλληλα υλοποιημένη μηχανή μπορούσε, από τα τέλη του 1990, να πραγματοποιήσει εξαντλητική αναζήτηση σε όλο το χώρο των κλειδιών μέσα σε 56 ώρες) και, από το 2004, έχει πλήρως αποσυρθεί

Ο βασικός και αντικειμενικός στόχος της κρυπτογραφίας είναι να δώσει τη δυνατότητα επικοινωνίας σε δύο πρόσωπα μέσα από ένα μη ασφαλές κανάλι με τέτοιο τρόπο έτσι ώστε ένα τρίτο, μη εξουσιοδοτημένο πρόσωπο (ένας αντίπαλος), να μην μπορεί να παρεμβληθεί στην επικοινωνία ή να κατανοήσει το περιεχόμενο των μηνυμάτων.

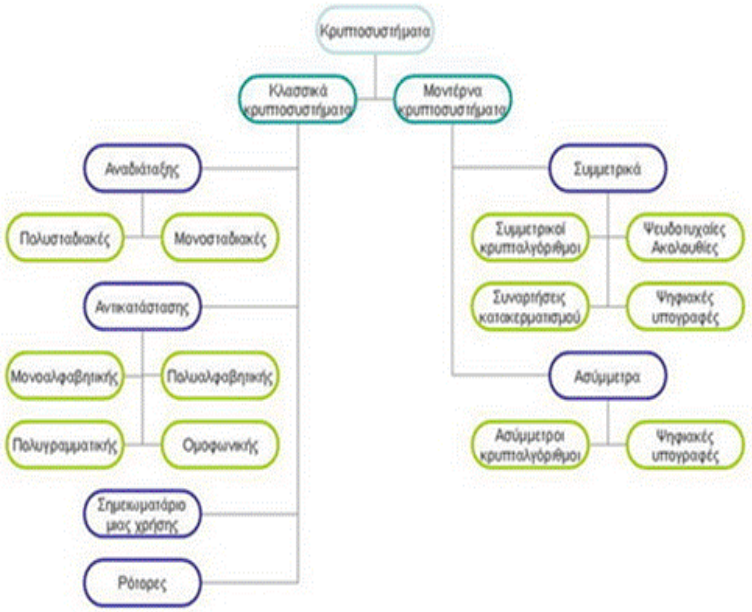
Έτσι, ένα **κρυπτοσύστημα** [6] ορίζεται ως το σύνολο διαδικασιών κρυπτογράφησης – αποκρυπτογράφησης και αποτελείται από μία πεντάδα (P,C,k,E,D) όπου:

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

Η συνάρτηση κρυπτογράφησης E δέχεται δύο παραμέτρους, μία από τον χώρο P και μία από τον χώρο k και παράγει μία ακολουθία η οποία ανήκει στον χώρο C.

Η συνάρτηση αποκρυπτογράφησης D δέχεται δύο παραμέτρους, τον χώρο C και τον χώρο k και παράγει μια ακολουθία που ανήκει στον χώρο P.

Όλα τα κρυπτοσυστήματα θα μπορούσαν να διακριθούν σε δύο μεγάλες κατηγορίες, τα **Κλασσικά Κρυπτοσυστήματα** και τα **Μοντέρνα Κρυπτοσυστήματα** (τα οποία, που είναι και τα πιο σημαντικά, αντιστοίχως κατηγοριοποιούνται σε συμμετρικά κρυπτοσυστήματα και ασύμμετρα κρυπτοσυστήματα).



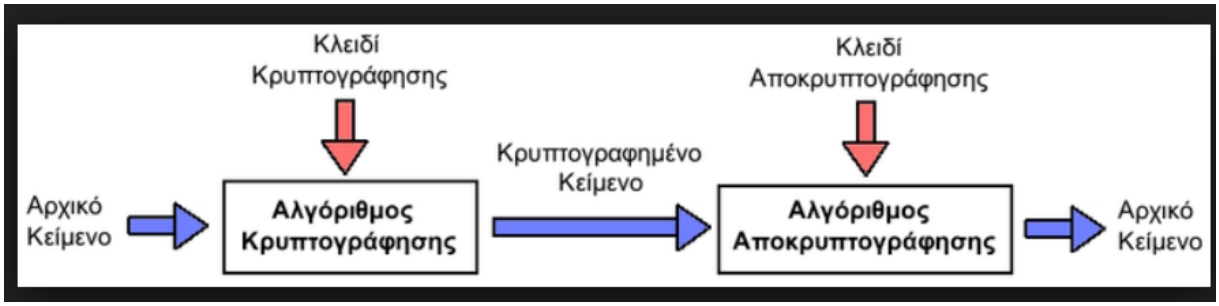
Εικόνα 3. Χάρτης κρυπτοσυστημάτων

Τα **κλασικά κρυπτοσυστήματα** [7] έχουν να κάνουν με την επεξεργασία γλωσσικών μηνυμάτων, δηλαδή μηνυμάτων που αποτελούνται από λέξεις όπου κάθε ψηφίο της λέξης αντιστοιχεί σε ένα από τα γράμματα αλφαβήτου, πχ 26 του αγγλικού. Για παράδειγμα η λέξη LAND μπορεί σε κάποιο σύστημα να κρυπτογραφείται LVNO. Η επεξεργασία πραγματοποιείται τόσο με αντικατάσταση του κάθε γράμματος με κάποιο άλλο γράμμα σύμφωνα με κάποια μέθοδο-κλειδί όσο και με αναδιάταξη στην σειρά-θέση στην οποία εμφανίζονται τα γράμματα σε μια λέξη. Τα κλασικά κρυπτοσυστήματα έχουν στην ουσία ιστορική πλέον αξία και απλά αναδεικνύουν συγκεκριμένες ιδιότητες που πρέπει να έχουν τα μοντέρνα συστήματα.

Στα **Μοντέρνα Κρυπτοσυστήματα** [7] όλοι οι καθιερωμένοι κρυπτογραφικοί αλγόριθμοι (συμμετρικοί ή ασύμμετροι), χρησιμοποιούν ένα κλειδί προκειμένου να παραμετροποιήσουν την κρυπτογράφηση αλλά και την αποκρυπτογράφηση ενός μηνύματος. Το κλειδί αυτό μπορεί να λαμβάνει μια τιμή, μέσα από ένα ευρύ φάσμα πιθανών τιμών, το οποίο ονομάζεται χώρος κλειδιών (keyspace). Σε αρκετούς αλγορίθμους το κλειδί που χρησιμοποιείται για την αποκρυπτογράφηση είναι διαφορετικό από το κλειδί της κρυπτογράφησης. Δεδομένου αυτής της παραδοχής, τα συστήματα κρυπτογράφησης μπορούν να χωριστούν σε δύο βασικές κατηγορίες: **συμμετρικά**, συμβατικά ή μυστικού κλειδιού (symmetric, conventional or secret key) όπου το κλειδί αποκρυπτογράφησης είναι το ίδιο με το κλειδί της κρυπτογράφησης, και **ασύμμετρα** ή δημοσίου κλειδιού (asymmetric or public key) όπου το κλειδί της αποκρυπτογράφησης είναι υπολογιστικά ανέφικτο να υπολογιστεί, γνωρίζοντας το κλειδί κρυπτογράφησης.

2.1 Συμμετρική Κρυπτογραφία

Στους συμμετρικούς αλγόριθμους το κλειδί κρυπτογράφησης ταυτίζεται με το κλειδί της αποκρυπτογράφησης. Ως εκ τούτου, αυτοί οι αλγόριθμοι χρειάζονται την εκ των προτέρων συμφωνία μεταξύ του αποστολέα και του παραλήπτη για το κλειδί που θα χρησιμοποιηθεί, για να μπορέσουν να επικοινωνήσουν με ασφάλεια. Η ασφάλεια των αλγόριθμων βασίζεται στην μυστικότητα αυτού του κλειδιού. Για όσο καιρό επιθυμούμε η επικοινωνία να παραμείνει μυστική, πρέπει και το κλειδί να παραμείνει μυστικό [8]. Μπορούμε να δούμε στη παρακάτω εικόνα τη διαδικασία της κρυπτογράφησης και αποκρυπτογράφησης



Εικόνα 4. Διαδικασία της κρυπτογράφησης και αποκρυπτογράφησης

Το κύριο πρόβλημα της συμμετρικής κρυπτογραφίας είναι η συνεννόηση του αποστολέα και του παραλήπτη στο κοινό μυστικό κλειδί που θα κρυπτογραφεί και αποκρυπτογραφεί όλη τη διακινούμενη πληροφορία, χωρίς κάποιον άλλο να λάβει γνώση αυτού. Πλεονέκτημα της είναι ότι είναι ταχύτερη από την ασύμμετρη κρυπτογραφία.

Για το παραπάνω πρόβλημα της συμμετρικής κρυπτογραφίας, προκύπτει η ανάγκη για συνεννόηση και ανταλλαγή του κλειδιού, χωρίς κάποιος τρίτος να μάθει για αυτό. Η μετάδοση μέσα από το Διαδίκτυο δεν είναι ασφαλής, γιατί οποιοσδήποτε γνωρίζει για τη συναλλαγή μπορεί να καταγράψει όλη την επικοινωνία μεταξύ αποστολέα και παραλήπτη και να αποκτήσει το κλειδί. Έπειτα, μπορεί να διαβάσει, να τροποποιήσει και να πλαστογραφήσει όλα τα μηνύματα που ανταλλάσσουν οι δύο ανυποψίαστοι χρήστες. Βέβαια, μπορούν να βασισθούν σε άλλο μέσο επικοινωνίας για τη μετάδοση του κλειδιού (π.χ. τηλεφωνία), αλλά ακόμα και έτσι δεν μπορεί να εξασφαλιστεί ότι κανείς δεν παρεμβάλλεται μεταξύ της γραμμής επικοινωνίας των χρηστών. Η ασύμμετρη κρυπτογραφία δίνει λύση σε αυτό το πρόβλημα, αφού σε καμία περίπτωση δεν μεταφέρονται στο δίκτυο οι εν λόγω ευαίσθητες πληροφορίες με την αρχική τους μορφή. Θα μιλήσουμε για ασύμμετρη κρυπτογραφία παρακάτω.

Υπάρχουν και περιπτώσεις, όπου η ασύμμετρη κρυπτογραφία δεν είναι απαραίτητη και η συμμετρική κρυπτογραφία από μόνη της είναι αρκετή. Τέτοιες περιπτώσεις είναι περιβάλλοντα κλειστά (π.χ. στρατός), που δεν έχουν σύνδεση με το Διαδίκτυο. Ένας υπολογιστής μπορεί να κρατά τα μυστικά κλειδιά των χρηστών που επιθυμούν να εξυπηρετηθούν από αυτόν, μια και δεν υπάρχει ο φόβος για κατάληψη της μηχανής από εξωτερικούς παράγοντες. Τέλος, στις περιπτώσεις που οι χρήστες μπορούν να συναντηθούν και να ανταλλάξουν τα κλειδιά ή όταν η κρυπτογράφηση χρησιμοποιείται για τοπική αποθήκευση κάποιων αρχείων, η ασύμμετρη κρυπτογραφία δεν είναι απαραίτητη.

Οι συμμετρικοί αλγόριθμοι μπορούν να ταξινομηθούν σε δύο υποκατηγορίες :

A. Αλγόριθμοι ροής (Stream ciphers)
B. Αλγόριθμοι τμήματος (block ciphers)

2.1.1 Αλγόριθμοι ροής (Stream ciphers)

Ο κρυπταλγόριθμος Ροής (stream cipher) είναι ένας τύπος αλγόριθμου συμμετρικής κρυπτογράφησης. Είναι εξαιρετικά ταχείς αλγόριθμοι, κατά πολύ ταχύτεροι από τους κώδικες τμήματος. Σε αντίθεση με τους κώδικες τμήματος που λειτουργούν με «μεγάλα» τμήματα δεδομένων (blocks), οι κώδικες ροής τυπικά λειτουργούν με μικρότερες μονάδες απλού κειμένου, συνήθως με bits.

Ένας κώδικας ροής παράγει μια ακολουθία από bits που χρησιμοποιείται σαν κλειδί και καλείται κλειδοροή (keystream). Η κρυπτογράφηση επιτυγχάνεται με τον συνδυασμό του keystream με το αρχικό μη κρυπτογραφημένο κείμενο, συνήθως μέσω της XOR πράξης. Η παραγωγή του keystream μπορεί να είναι ανεξάρτητη του αρχικού κειμένου και του κρυπτογραφήματος (συγχρονισμένοι κώδικες ροής (synchronous stream cipher)) ή μπορεί να εξαρτάται από αυτά (ασύγχρονοι κώδικες ροής (self-synchronizing stream cipher)).

2.1.2 Αλγόριθμοι τμήματος (block ciphers)

Ο κρυπταλγόριθμος Τμήματος (Block Cipher) είναι ένας τύπος αλγόριθμου συμμετρικής κρυπτογράφησης που μετατρέπει ένα τμήμα (block) μη κρυπτογραφημένου καθορισμένου μήκους κειμένου (plaintext), σε τμήμα (block) κρυπτοκειμένου (ciphertext). Αυτός ο μετασχηματισμός πραγματοποιείται με την βοήθεια ενός μυστικού κλειδιού που δίνεται από το χρήστη. Η αποκρυπτογράφηση γίνεται με την εφαρμογή του αντίστροφου μετασχηματισμού στο κρυπτογραφημένο κείμενο χρησιμοποιώντας το ίδιο μυστικό κλειδί. Το καθορισμένο μήκος καλείται

μήκος τμήματος (block size) (π.χ. 64, 128, 196... bits). Κάθε τμήμα δίνει διαφορετικό κρυπτογραφημένο κείμενο (ciphertext).

Οι κρυπταλγόριθμοι τμήματος (block ciphers) λειτουργούν επαναληπτικά, κρυπτογραφώντας ένα τμήμα διαδοχικά αρκετές φορές. Σε κάθε γύρο, ο ίδιος μετασχηματισμός εφαρμόζεται στα δεδομένα χρησιμοποιώντας ένα υποκλειδί (subkey). Το σύνολο των υποκλειδιών προέρχεται από το μυστικό κλειδί που παρείχε ο χρήστης, με ειδική συνάρτηση. Το σύνολο των υποκλειδιών καλείται σχεδιασμός κλειδιών (key schedule).

Ο αλγόριθμος DES (Data Encryption Standard) ήταν ο πρώτος καθολικά αποδεκτός block cipher. Αντιπροσωπεύει την τυποποίηση Federal Information Processing Standard (FIPS) 46-1 που επίσης περιγράφει τον Data Encryption Algorithm (DEA). Αρχικά αναπτύχθηκε από την IBM, ενώ σημαντικό ρόλο στην ανάπτυξη του έπαιξε η NSA και το National Institute of Standards and Technology (NIST).

Ο DES είναι block cipher και πιο συγκεκριμένα δομής Feistel (πρόκειται για μία συγκεκριμένη δομή κρυπταλγορίθμων τμήματος, που έχουν την ιδιότητα ότι το κύκλωμα αποκρυπτογράφησης ταυτίζεται με το κύκλωμα κρυπτογράφησης), , με μέγεθος block 64 bit. Χρησιμοποιεί κλειδί 64 bits από τα οποία τα 8 αποτελούν bits ισοτιμίας – άρα, από πλευράς ασφάλειας, το κλειδί έχει μέγεθος 56 bits. Όταν χρησιμοποιείται για την επικοινωνία, αποστολέας και παραλήπτης μοιράζονται το ίδιο κλειδί. Επίσης, όπως και κάθε άλλος κρυπταλγόριθμος τμήματος, μπορεί να χρησιμοποιηθεί για κρυπτογράφηση αρχείων αποθηκευμένα σε σκληρό δίσκο σε περιβάλλοντα ενός χρήστη. Για την διανομή των κλειδιών σε περιβάλλον πολλών χρηστών, συνδυάζεται με ασύμμετρο κρυπτοσύστημα.

Ο 3DES είναι μια παραλλαγή του DES όπου το μήνυμα κρυπτογραφείται και αποκρυπτογραφείται διαδοχικά με διαφορετικά κλειδιά για την ενίσχυση του βασικού αλγόριθμου. Ουσιαστικά πρόκειται για μία τριπλή εφαρμογή του DES, έτσι ώστε να διατηρηθούν οι καλές κρυπτογραφικές ιδιότητες του DES αλλά να αυξηθεί το μέγεθος του κλειδιού (σε 3 x 56 = 168 bits).

Ο DESX είναι μια άλλη παραλλαγή του DES. Η διαφορά του DES και του DESX είναι ότι η είσοδος στο DESX περνάει από μια XOR πράξη με ένα επιπλέον κλειδί 64 bits και ομοίως η έξοδος της κρυπτογράφησης.

Ο DES δεν θεωρείται πια ασφαλής αλγόριθμος και έχει αντικατασταθεί από τον AES, για τον οποίο θα μιλήσουμε παρακάτω.

2.1.3 One Time Pad. Ο «απόλυτος» αλγόριθμος!

Οι αλγόριθμοι ροής που αναφέραμε πιο πριν, βασίζονται στις θεωρητικές ιδιότητες ενός «one-time pad». Τα one-time pads (καμιά φορά καλούνται και Vernam κρυπτοσυστήματα) είναι τα κρυπτοσυστήματα που χρησιμοποιούν μια ακολουθία bits (keystream) που παράγεται τελείως στην τύχη. Η ακολουθία των bits είναι του ίδιου μήκους με το μη κρυπτογραφημένο κείμενο και συνδυάζεται μέσω μιας XOR πράξης με το αυτό για την παραγωγή του κρυπτογραφήματος. Επειδή η ακολουθία των bits είναι τελείως τυχαία και είναι του ίδιου μήκους με το αρχικό κείμενο, η εύρεση του κειμένου είναι αδύνατη ακόμα και με τη διάθεση τεράστιας υπολογιστικής ισχύος – εφόσον το κλειδί χρησιμοποιείται μόνο μία φορά, δηλαδή δεν επαναχρησιμοποιείται. Ένα τέτοιο κρυπτοσύστημα προσφέρει τέλεια μυστικότητα και ασφάλεια και έχει χρησιμοποιηθεί σε μεγάλη κλίμακα σε καιρό πολέμου για τη διασφάλιση διπλωματικών καναλιών. Το γεγονός, όμως, ότι το μυστικό κλειδί (δηλαδή το keystream), που χρησιμοποιείται μόνο μία φορά, είναι του ίδιου μήκους με το μήνυμα, εισάγει σημαντικό πρόβλημα στη διαχείριση του κλειδιού. Φανταστείτε να θέλαμε να στείλουμε με «one-time pad» ένα video ποιότητας HD. Και να έπρεπε να το επαναλάβουμε εκατοντάδες φορές! Παρ' όλη την ασφάλεια που προσφέρει, ο «one-time pad» δεν μπορεί να εφαρμοστεί στην πράξη.

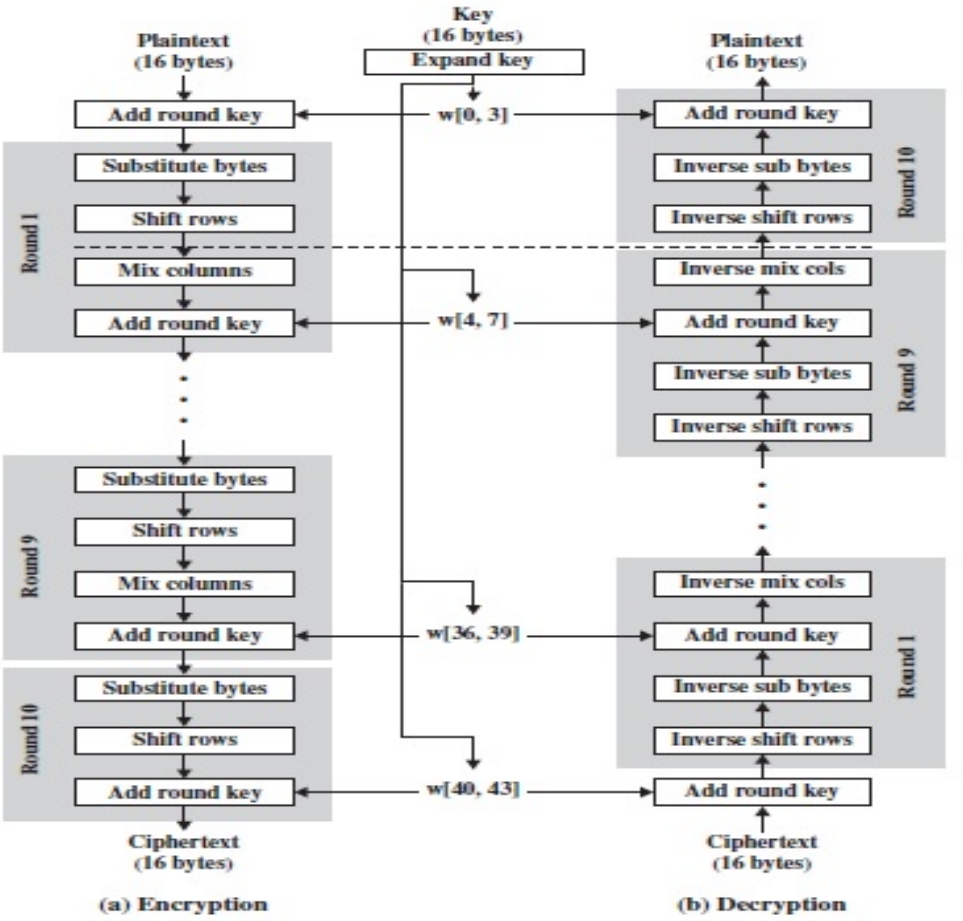
2.1.4 Ο αλγόριθμος AES

Το κυριότερο μειονέκτημα στην υλοποίηση του 3DES αποτελεί το γεγονός ότι ο αλγόριθμος είναι σχετικά αργός σε υλοποιήσεις με χρήσεις λογισμικού (software). Το σύστημα DES σχεδιάστηκε τη δεκαετία του ’70 για υλοποίηση με χρήση υλικού αλλά δε παράγει αποδοτικό κώδικα λογισμικού. Ο 3DES, είναι πολύ πιο βραδύτερος από τον DES καθώς οι γύροι που περιλαμβάνει είναι τρεις φορές περισσότεροι. Ένα άλλο μειονέκτημα είναι η απαίτηση που εμφανίζουν οι DES και 3DES να χρησιμοποιούν τμήματα μεγέθους 64 bit, το οποίο μέγεθος, για γενικότερους λόγους αποδοτικότητας και ασφάλειας, είναι επιθυμητό να είναι μεγαλύτερο. Επομένως, καταλήγουμε στο συμπέρασμα ότι ο 3DES δεν μπορεί να θεωρηθεί αποτελεσματικός.

Βάση των παραπάνω αδυναμιών του DES και 3DES, ο NIST προκήρυξε διαγωνισμό πρότασης για τον AES. Ο καινούργιος αλγόριθμος θα έπρεπε να είναι το ίδιο αποδοτικός και ισχυρός, αλλά επιπλέον θα έπρεπε να έχει λιγότερες απαιτήσεις σε μνήμη και λιγότερες απαιτήσεις σε υλικό και λογισμικό για την υλοποίησή του. Τελικά νικητής του διαγωνισμού αναδείχθηκε η πρόταση των Βέλγων κρυπτογράφων Δρ. Joan Daemen και του Δρ. Vincent Rijmen και το τελικό πρότυπο δημοσιεύτηκε το 2001 στο Federal Information Processing Standards Publications 197 [5]

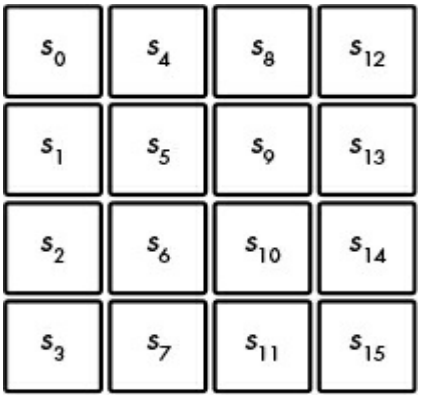
Ο AES κρυπτογραφεί τμήματα των 128-bit και έχει τη δυνατότητα επιλογής κλειδιού κρυπτογράφησης μεταξύ των 128-bit, 192-bit και 256-bit. Το τμήμα των 128bit επιλέχθηκε γιατί κάνει την κρυπτογράφηση πιο γρήγορη και η διαφορά σε θέματα ασφάλειας με το τμήμα των 256bit, για τις περισσότερες εφαρμογές, είναι πρακτικά αμελητέα.

Χρησιμοποιεί πίνακες από bytes, μεγέθους 4x4, όπως φαίνεται στην παρακάτω εικόνα , για να κάνει κάποιες επαναλήψεις, των οποίων ο αριθμός τους εξαρτάται από το μέγεθος του κλειδιού. Το κλειδί που χρησιμοποιείται σε κάθε επανάληψη, προκύπτει από το αρχικό κλειδί. Σε κάθε επανάληψη, συμβαίνει αντικατάσταση των bytes με τη χρήση S-box, ολίσθηση γραμμών, ανάμειξη δεδομένων σε κάθε στήλη του πίνακα, και τέλος, τη πράξη XOR του πίνακα με το κλειδί [5] [9]



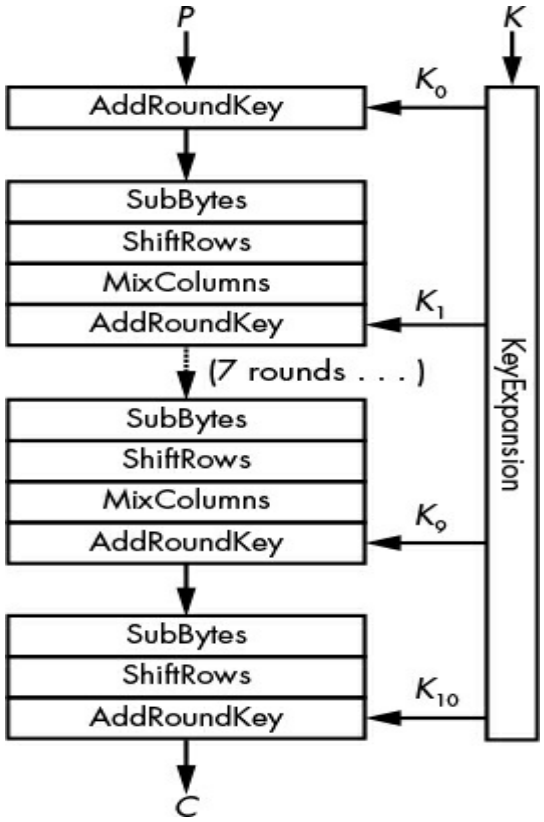
Εικόνα 5. Κρυπτογράφηση και αποκρυπτογράφηση του αλγορίθμου AES

Ενώ λοιπόν άλλοι αλγόριθμοι χρησιμοποιούν ανεξάρτητα bits ή 64-bit words, ο AES διαχειρίζεται bytes. Ένα κείμενο μεγέθους 16-byte το θεωρεί σαν έναν δυσδιάστατο πίνακα από bytes (s = s0, s1, . . . , s15), όπως φαίνεται και στην παρακάτω εικόνα (χρησιμοποιείται το γράμμα «s», γιατί αυτός ο πίνακας στην αγγλική ορολογία ονομάζεται «internal state», ή απλά «state».). Ο AES μετατρέπει τα bytes, δηλαδή τις γραμμές και τις στήλες του πίνακα και τελικά παράγει το αποτέλεσμα, το οποίο είναι το κρυπτοκείμενο.



Εικόνα 6. Η εσωτερική δομή (state) του AES ως πίνακας 4 × 4 από bytes

Για να κάνει την μετατροπή, ο AES χρησιμοποιεί ένα δίκτυο αντικατάστασης-μετάθεσης SPN (substitution–permutation network). Η δομή του φαίνεται στην παρακάτω εικόνα με 10 γύρους για 128-bit κλειδί, 12 για 192-bit κλειδί και 14 για 256-bit κλειδί.

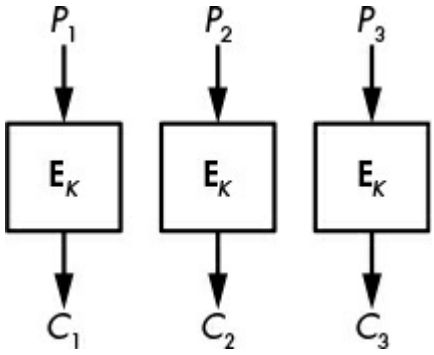


Εικόνα 7. Η εσωτερική λειτουργία του AES

Οι διαθέσιμοι τρόποι λειτουργίας του AES (οι οποίοι ουσιαστικά μπορούν να εφαρμοστούν σε οποιονδήποτε κρυπταλγόριθμο τμήματος) είναι οι παρακάτω:

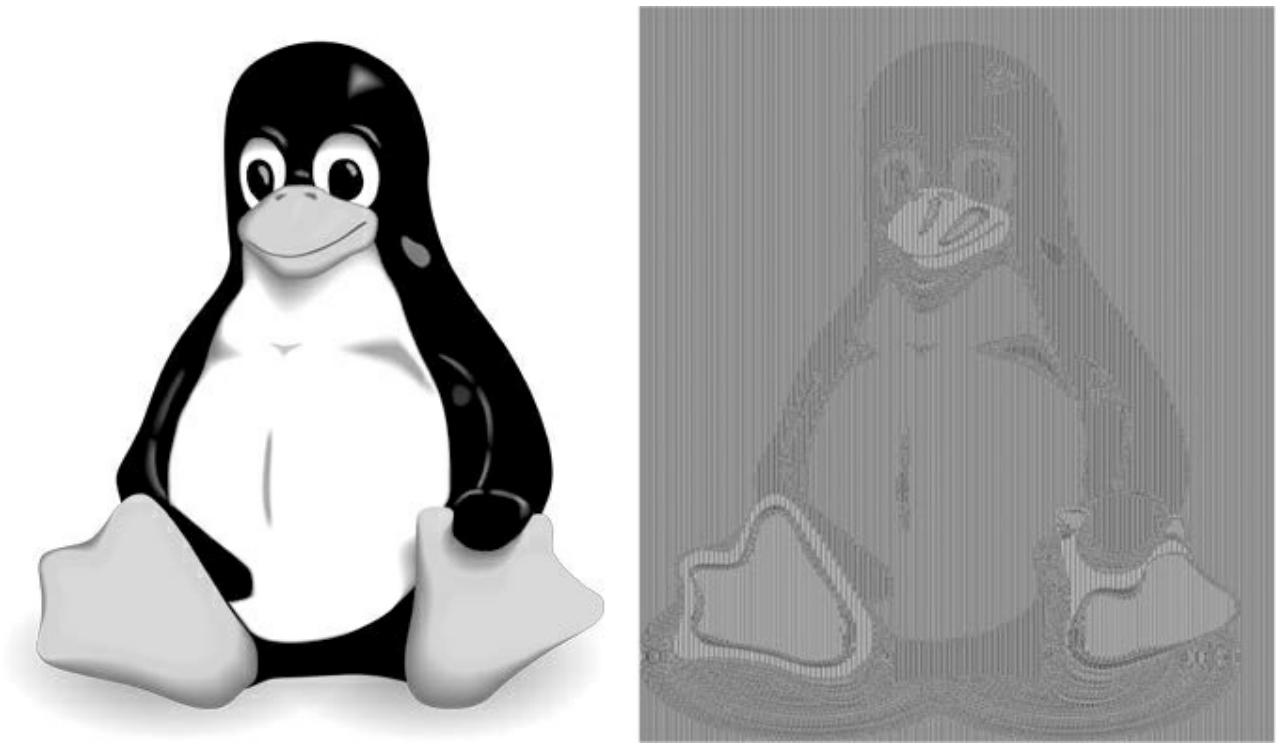
Electronic Codebook (ECB) Mode:

Είναι ο απλούστερος τρόπος λειτουργίας. Για αρκετούς κρυπτογράφους δεν θεωρείται καν ως «κανονικός» τρόπος! Σε ECB mode, ο AES παίρνει τα μπλοκ μηνύματος P1, P2, . . . , PN και τα επεξεργάζεται ανεξάρτητα το καθένα. Έτσι δημιουργεί τα C1 = E(K, P1), C2 = E(K, P2), και ου το καθεξής. Σχηματικά ο τρόπος λειτουργίας ECB φαίνεται στο παρακάτω σχήμα. Είναι ο πιο απλός τρόπος λειτουργίας αλλά και ο λιγότερο ασφαλής. Όταν χρησιμοποιείτε τον AES για σοβαρές εφαρμογές μη χρησιμοποιείτε ποτέ το ECB mode !



Εικόνα 8. Ο τρόπος λειτουργίας ECB

Υπάρχει ένα πασίγνωστο παράδειγμα που δείχνει οπτικά ότι ο AES ECB είναι ακατάλληλος για χρήση. Είναι η κρυπτογράφηση της εικόνας του Tux , της μασκότ του Linux και η οπτικοποίηση ξανά του παραγόμενου κρυπτοκειμένου. Όπως βλέπουμε στην παρακάτω εικόνα, το σχήμα της εικόνας δεν άλλαξε καθόλου, καθώς τελικά , η μόνη αλλαγή που έγινε από τον αλγόριθμο, ήταν η τιμή του γκρι χρώματος!.



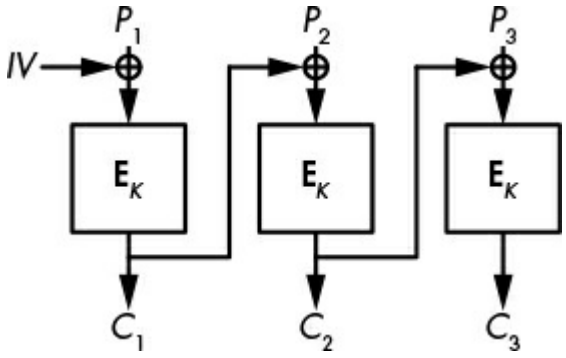
Εικόνα 9. Η αρχική εικόνα (αριστερά) και η κρυπτογραφημένη με ECB (δεξιά)

Επίσης, σε ECB mode, τα ίδια μπλοκ κρυπτοκειμένου, αντιστοιχούν σε ολόιδια μπλοκ αρχικού μηνύματος. Άρα είναι μεγάλη ευπάθεια για σκοπούς κρυπτανάλυσης και οι επιτιθέμενοι μπορούν να τη χρησιμοποιήσουν με πολλούς τρόπους!

Τέλος ο AES ECB διαχειρίζεται μόνο μπλοκ αρχικού κειμένου που είναι πολλαπλάσιο των 16 bytes: διαφορετικά, το τελευταίο block χρειάζεται συμπλήρωση (padding). Σε κάποιους από τους επόμενους τρόπους λειτουργίας, αυτό μπορεί να ξεπεραστεί με διάφορες τεχνικές.

The Cipher Block Chaining (CBC) Mode:

Ο τρόπος λειτουργίας Cipher block chaining (CBC) είναι παρόμοιος με τον ECB αλλά με μια μικρή διαφορά. Αντί να κρυπτογραφεί το μπλοκ Pi σαν Ci = E(K,Pi), ο CBC θέτει Ci = E(K, Pi $\oplus$ Ci − 1), όπου το Ci − 1 είναι το προηγούμενο παραγμένο κρυπτομπλόκ. Έτσι το προηγούμενο κρυπτομπλόκ συνδέεται με το επόμενο. Για το πρώτο τμήμα κρυπτοκειμένου ο CBC παίρνει ένα τυχαίο μπλοκ, το οποίο ονομάζεται διάνυσμα αρχικοποίησης «initialization vector» (IV). Ο τρόπος αυτός φαίνεται σχηματικά στην παρακάτω εικόνα.

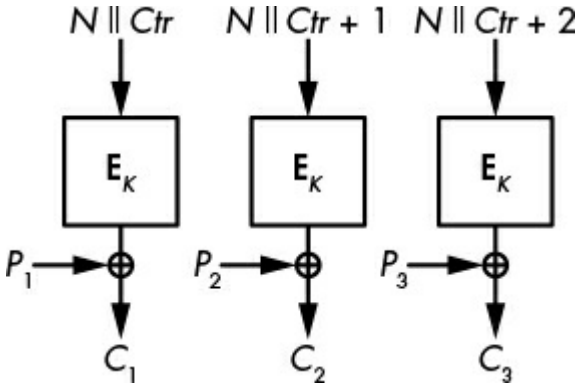


Εικόνα 10. Ο τρόπος λειτουργίας CBC

Σε πολλές υλοποιήσεις του αλγορίθμου χρησιμοποιείται ένα προκαθορισμένο «IV» αντί για κάποιο τυχαίο. Έτσι πολλά κρυπτοκείμενα που αρχίζουν με το ίδιο μπλοκ, θα έχουν και το ίδιο αρχικό μπλοκ κειμένου. Ένας κρυπταναλυτής λοιπόν, μπορεί να δει ότι δύο κρυπτοκείμενα αρχίζουν από το ίδιο μπλοκ κειμένου, ακόμα και αν τα κρυπτοκείμενα είναι διαφορετικά μεταξύ τους.

The Counter (CTR) Mode:

Ο τρόπος λειτουργίας CTR ελαχιστοποιεί όλες τις ευπάθειες των προηγούμενων τρόπων και κυρίως δυσκολεύει πάρα πολύ την κρυπτανάλυση που στηρίζεται στη «κλοπή» του κρυπτοκειμένου. Ο AES CTR δεν είναι ακριβώς ένας «block cipher». Μετατρέπει τον AES σε «stream cipher» ο οποίος διαχειρίζεται bits και όχι blocks από bytes. Ο τρόπος λειτουργίας φαίνεται στην παρακάτω εικόνα.



Εικόνα 11. Ο τρόπος λειτουργίας CTR

Ο AES CTR κρυπτογραφεί τα μπλοκ του κειμένου χρησιμοποιώντας έναν «counter» και έναν αρχικό αριθμό που ονομάζεται «nonce» (number used only once). Ο «counter» είναι ένας ακέραιος που αυξάνει κατά ένα σε κάθε μπλοκ. Δυο μπλοκ στο ίδιο μήνυμα δεν μπορούν να έχουν «counter» με την ίδια τιμή. Ο αριθμός «nonce» χρησιμοποιείται μόνο μία φορά και είναι ο ίδιος για όλα τα μπλοκ του ίδιου μηνύματος , αλλά αλλάζει τιμή για διαφορετικά μηνύματα.

Σε αυτή τη λειτουργία, ο αλγόριθμος κρυπτογραφεί το «nonce», το N και το «counter» και το «stream» που παράγεται γίνεται XOR με το μπλοκ του μηνύματος. Η αποκρυπτογράφηση είναι ίδια, οπότε χρησιμοποιείται ο ίδιος αλγόριθμος.

Ένα βασικό πλεονέκτημα του AES CTR είναι η ταχύτητα. Είναι γρηγορότερος από τους άλλους τρόπους λειτουργίας. Κατά τη διάρκεια της αποκρυπτογράφησης μπορούμε να εφαρμόσουμε παράλληλη επεξεργασία, επειδή για την αποκρυπτογράφηση ενός οποιουδήποτε μπλοκ δεν χρειάζεται να έχει ολοκληρωθεί η αποκρυπτογράφηση των προηγούμενων (κάτι που δεν ισχύει στον CBC τρόπο λειτουργίας), οπότε και μπορούμε να το εκμεταλλευτούμε για να επιτύχουμε παραλληλισμό και να αυξήσουμε δραστικά το χρόνο αποκρυπτογράφησης.

Αυτός είναι και ο βασικός λόγος που **επιλέξαμε τον AES CTR για την σύγκριση με τον SATURNIN** σε περιβάλλον IoT (Internet of Things). Θα εξετάσουμε διεξοδικά το περιβάλλον στο οποίο έγινε η δοκιμή απόδοσης, στα παρακάτω κεφάλαια. Υπάρχουν και άλλοι τρόποι λειτουργίας των κρυπταλγορίθμων τμήματος, οι οποίοι δεν θα περιγράφουν στην παρούσα διατριβή αφού εκ φεύγουν του αντικειμένου της.

2.1.5 Ασφάλεια στον AES

Είναι ασφαλής ο AES;

Ο AES είναι όσο ασφαλής θα μπορούσε ποτέ να είναι ένας block cipher. Βασικά, ο AES είναι ασφαλής επειδή όλα τα bit εξόδου εξαρτώνται από όλα τα bit εισόδου με κάποιο πολύπλοκο, ψευδοτυχαίο τρόπο. Για να το επιτύχουν αυτό, οι σχεδιαστές του AES επέλεξαν προσεκτικά κάθε συστατικό του για συγκεκριμένο λόγο, π.χ. τα MixColumns για τις μέγιστες ιδιότητες διάχυσης και τα SubBytes για τη βέλτιστη μη γραμμικότητα. Ως τώρα τα παραπάνω έχουν δείξει ότι αυτή η σύνθεση προστατεύει τον AES από πολλές κατηγορίες κρυπτοαναλυτικών επιθέσεων.

Αλλά δεν υπάρχει μαθηματική απόδειξη ότι το AES είναι ανθεκτικός σε όλες τις πιθανές επιθέσεις. Πρώτον, δεν γνωρίζουμε ποιες είναι όλες οι πιθανές επιθέσεις και δεν ξέρουμε πάντα πώς να αποδείξουμε ότι ένας αλγόριθμος είναι ασφαλής έναντι μιας συγκεκριμένης επίθεσης.

Μετά από περισσότερα από 15 χρόνια και εκατοντάδες ερευνητικές δημοσιεύσεις, η θεωρητική ασφάλεια του AES έχει ελάχιστα χαθεί. Το 2011 οι κρυπταναλυτές βρήκαν έναν τρόπο να ανακτήσουν ένα κλειδί AES-128 εκτελώντας περίπου 2^{126} λειτουργίες αντί για 2^{128}. Αλλά αυτή η «επίθεση»

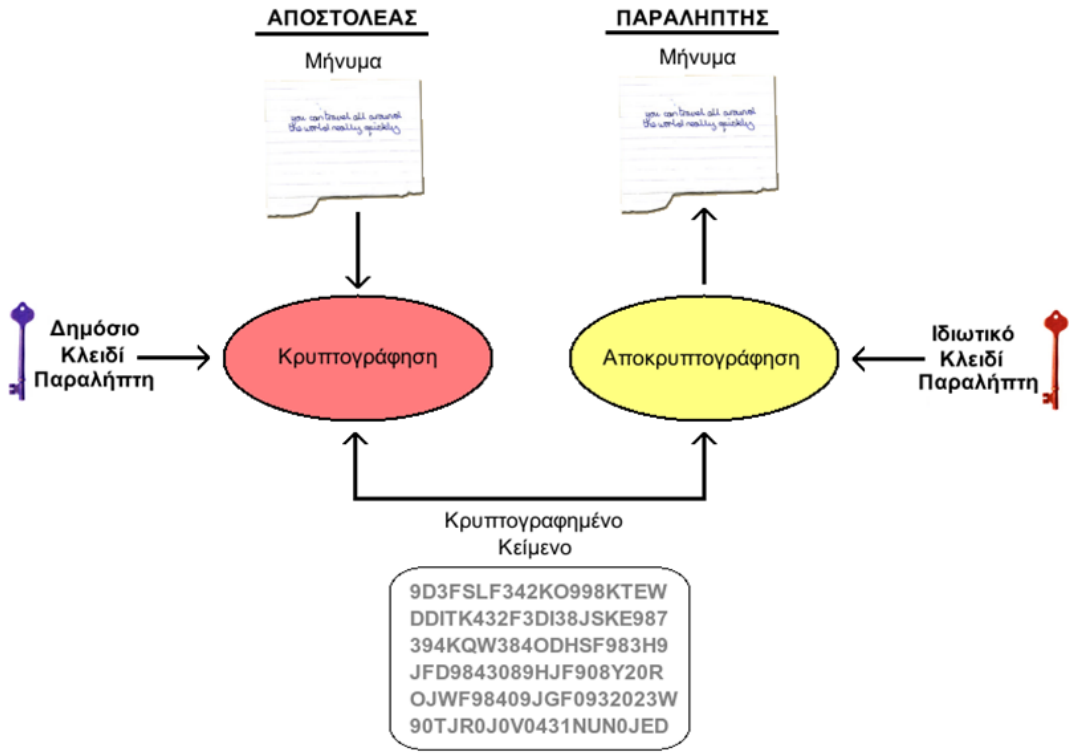
απαιτούσε μια πολύ μεγάλη ποσότητα ζευγών απλού κειμένου – κρυπτοκειμένου για ένα αρχικό μήνυμα μεγέθους 288 bit. Άρα ο αλγόριθμος συνεχίζει να είναι ισχυρός!

Ακόμα και με κβαντική κρυπτανάλυση, ένα κλειδί μεγέθους 256bit συνεχίζει να διατηρεί τον αλγόριθμο ασφαλή. Ο Saturnin, στον οποίον επικεντρώνεται η έρευνά μας στη συνέχεια ως μελέτη περίπτωσης, στηρίζεται στον AES και κληρονομεί όλα τα χαρακτηριστικά που τον κάνουν ασφαλή.

2.2 Ασύμμετρη Κρυπτογραφία

Η μεγαλύτερη αδυναμία των περισσότερων κρυπτοσυστημάτων ήταν πάντα η διανομή των κλειδιών. Οι ασύμμετροι αλγόριθμοι είναι σχεδιασμένοι έτσι ώστε το κλειδί που χρησιμοποιούν για την κρυπτογράφηση να είναι διαφορετικό από το κλειδί για την αποκρυπτογράφηση και το κλειδί της αποκρυπτογράφησης δεν μπορεί να προκύψει από εκείνο της κρυπτογράφησης.

Οι αλγόριθμοι αυτοί ονομάζονται και δημοσίου κλειδιού γιατί το κλειδί της κρυπτογράφησης μπορεί να δημοσιοποιηθεί. Ο καθένας μπορεί να κρυπτογραφήσει ένα μήνυμα με το δημόσιο κλειδί του παραλήπτη αλλά μόνο ο κάτοχος του ιδιωτικού κλειδιού –δηλαδή, ο παραλήπτης - μπορεί να το αποκρυπτογραφήσει. Παραδείγματα ασύμμετρων αλγορίθμων κρυπτογράφησης είναι ο RSA, οι αλγόριθμοι ελλειπτικών καμπυλών, ο El Gamal και άλλοι. [10]



Εικόνα 12. Αναπαράσταση ενός συστήματος δημοσίου κλειδιού

Ένα πλεονέκτημα των ασύμμετρων κρυπτοσυστημάτων είναι ότι μπορούν να παρέχουν ψηφιακές υπογραφές που δεν μπορούν να αποκηρυχθούν από την πηγή τους. Η πιστοποίηση ταυτότητας μέσω συμμετρικής κρυπτογράφησης απαιτεί την κοινή χρήση του ίδιου κλειδιού και πολλές φορές τα κλειδιά αποθηκεύονται σε υπολογιστές που κινδυνεύουν από εξωτερικές επιθέσεις. Σαν αποτέλεσμα, ο αποστολέας μπορεί να αποκηρύξει ένα πρωτύτερα υπογεγραμμένο μήνυμα, υποστηρίζοντας ότι το μυστικό κλειδί είχε κατά κάποιον τρόπο αποκαλυφθεί. Στην ασύμμετρη κρυπτογραφία δεν επιτρέπεται κάτι τέτοιο, αφού κάθε χρήστης έχει αποκλειστική γνώση του ιδιωτικού του κλειδιού και είναι δικιά του ευθύνη η φύλαξή του

Κυριότερο μειονέκτημα της ασύμμετρης κρυπτογραφίας είναι η ταχύτητα. Κατά κανόνα, οι διαδικασίες κρυπτογράφησης και πιστοποίησης ταυτότητας με συμμετρικό κλειδί είναι σημαντικά ταχύτερες από την κρυπτογράφηση και ψηφιακή υπογραφή με ζεύγος ασύμμετρων κλειδιών.

Ένα άλλο μειονέκτημα της ασύμμετρης κρυπτογραφίας προκύπτει από την ανάγκη για πιστοποίηση και επαλήθευση των δημόσιων κλειδιών από οργανισμούς Πιστοποίησης (Certificate Authority) ώστε να διασφαλίζεται η κατοχή των νόμιμων χρηστών. Όταν κάποιος επιτήδειος κατορθώσει και ξεγελάσει τον οργανισμό, μπορεί να συνδέσει το όνομά του με το δημόσιο κλειδί ενός νόμιμου χρήστη και να προσποιείται την ταυτότητα αυτού του νόμιμου χρήστη

Όπως θα δούμε παρακάτω η ασύμμετρη κρυπτογραφία είναι μέρος του πρωτοκόλλου TLS το οποίο και χρησιμοποιείται ευρέως στο διαδίκτυο. Με την έλευση όμως των κβαντικών υπολογιστών οι γνωστοί μέχρι τώρα αλγόριθμοι ασύμμετρης κρυπτογραφίας θα είναι μη ασφαλείς.

2.3 Ακεραιότητα και Αυθεντικότητα Μηνυμάτων

Κατά τη μεταφορά δεδομένων με τη μορφή μηνυμάτων στο Διαδίκτυο, κρίσιμο ζητούμενο αποτελεί η ύπαρξη μηχανισμών για την επιβεβαίωση της ακεραιότητας και αυθεντικότητας του κάθε μηνύματος στο σημείο παραλαβής του. Σημαντικό εργαλείο σε αυτή την κατεύθυνση αποτελούν οι μηχανισμοί ελέγχου ακεραιότητας, με τη χρήση κρυπτογραφικών συναρτήσεων κατακερματισμού (cryptographic hash functions), καθώς και ελέγχου αυθεντικότητας του αποστολέα του μηνύματος με πρόσθετες τεχνικές.

2.3.1 Συναρτήσεις Κατακερματισμού

Ο όρος συνάρτηση κατακερματισμού (hash function) h υποδηλώνει ένα μετασχηματισμό που παίρνει ως είσοδο ένα μήνυμα m οποιουδήποτε μήκους και επιστρέφει στην έξοδο μία ακολουθία χαρακτήρων h(m) περιορισμένου μήκους που καλείται τιμή κατακερματισμού (hash value). Οι συναρτήσεις κατακερματισμού είναι συναρτήσεις με τις εξής ιδιότητες:

- Η είσοδος είναι οποιουδήποτε μήκους.
- Η έξοδος έχει περιορισμένο, σταθερό, μήκος.
- Δεδομένου του m, ο υπολογισμός του h(m) είναι εύκολος.
- Η h είναι μη αντιστρέψιμη.
- Η h δεν είναι αμφιμονοσήμαντη (ένα προς ένα συνάρτηση).
- Αδύναμη αντίσταση σε σύγκρουση (weak collision resistance ή second preimage resistance): Για γνωστό m να είναι υπολογιστικά μη εφικτό να βρεθεί m' ≠ m ώστε να ισχύει H(m) = H(m').
- Ισχυρή αντίσταση σε σύγκρουση (strong collision resistance): Δεν είναι υπολογιστικά εφικτό να βρεθούν δύο διαφορετικές είσοδοι m, m' που να δίνουν την ίδια έξοδο, δηλαδή H(m)=H(m').

Η τιμή κατακερματισμού παρουσιάζει συνοπτικά το μεγαλύτερο μήνυμα ή έγγραφο, για αυτό καλείται και σύνοψη μηνύματος (message digest). Μπορούμε να φανταστούμε τη σύνοψη του μηνύματος σαν "ψηφιακό αποτύπωμα" ("digital fingerprint") του εγγράφου. Παραδείγματα γνωστών συναρτήσεων κατακερματισμού είναι οι MD2, MD5 (μη ασφαλείς σήμερα) και οι οικογένειες αλγορίθμων SHA-1 (επίσης μη ασφαλής πια), SHA-2 και SHA-3.

2.3.2 Κώδικας Αυθεντικοποίησης Μηνυμάτων - MAC

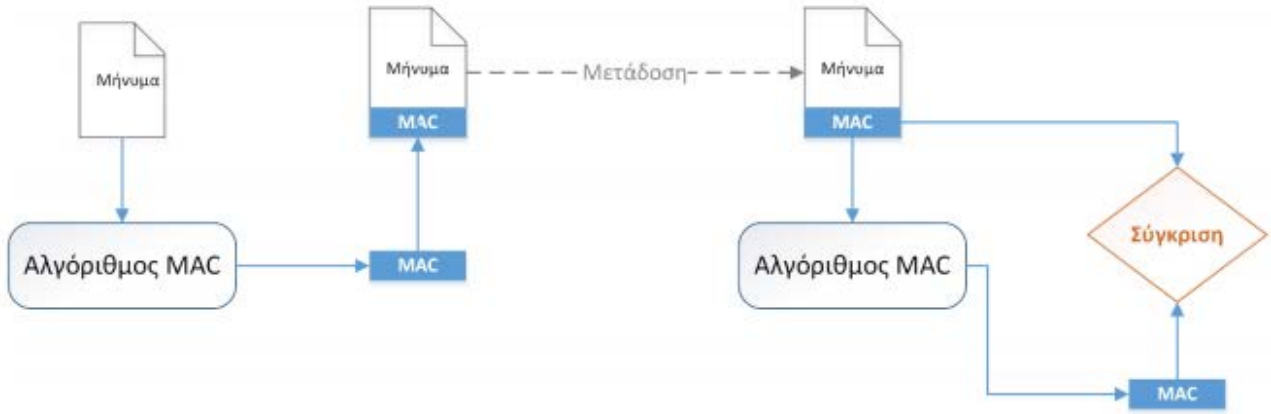
Ο Κώδικας Αυθεντικοποίησης Μηνύματος (Message Authentication Code - MAC) παρέχει αυθεντικοποίηση της πηγής προέλευσης (data origin authentication) και προστασία ακεραιότητας του ίδιου του μηνύματος. Για τον υπολογισμό του χρησιμοποιούνται δυο είσοδοι:

- Το μήνυμα
- Ένα μυστικό κλειδί κρυπτογράφησης

Με απλά λόγια, έχει όλες τις ιδιότητες μίας συνάρτησης κατακερματισμού, αλλά επιπροσθέτως υπεισέρχεται ένα μυστικό κλειδί για τον υπολογισμό.

Κατά τη διαδικασία αυθεντικοποίησης, ο παραλήπτης χρησιμοποιεί το κοινό μυστικό κλειδί που είχε χρησιμοποιήσει και ο αποστολέας κατά τον υπολογισμό του κώδικα MAC. Πιο συγκεκριμένα, χρησιμοποιείται ένα κλειδί και μια κρυπτογραφική συνάρτηση για τον υπολογισμό της τιμής-ελέγχου (MAC) των δεδομένων, που κατόπιν στέλνεται μαζί με τα δεδομένα.

Ο κώδικας MAC προϋποθέτει το διαμοιρασμό μεταξύ δύο πλευρών ενός μυστικού κλειδιού, προκειμένου να αυθεντικοποιήσουν την πηγή προέλευσης (origin) των μεταξύ τους μεταδιδόμενων δεδομένων. Η αυθεντικότητα της πηγής προέλευσης των δεδομένων, την οποία εγγυάται η συνάρτηση MAC, απορρέει από το γεγονός ότι δεν είναι δυνατό να αναπαραχθεί η ίδια συνόψιση χωρίς τη γνώση του κλειδιού. Ένας ενδιάμεσος ο οποίος υποκλέπτει το μήνυμα δεν θα μπορέσει να τροποποιήσει τα δεδομένα του μηνύματος και μετά να παράγει σωστό κώδικα MAC. Στην παρακάτω εικόνα, απεικονίζεται ο τρόπος αξιοποίησης του μηχανισμού MAC κατά τη μετάδοση μηνυμάτων.



Εικόνα 13. Αξιοποίηση μηχανισμού MAC

Ο πιο γνωστός MAC είναι ο λεγόμενος αλγόριθμος HMAC, ο οποίος αποδεικνύεται ότι είναι ασφαλής όσο η υποκείμενη συνάρτηση κατακερματισμού είναι ασφαλής (οπότε εάν ο HMAC χρησιμοποιεί είτε τον SHA-2 είτε τον SHA-3 είναι ασφαλής).

Κεφάλαιο 3
Διαδίκτυο των «πραγμάτων»

Ο όρος «Διαδίκτυο των πραγμάτων» (Internet of Things - IoT) [11]χρησιμοποιήθηκε για πρώτη φορά το 1999 από τον Βρετανό πρωτοπόρο στην τεχνολογία, Kevin Ashton, για να περιγράψει ένα σύστημα στο οποίο τα αντικείμενα του φυσικού κόσμου θα μπορούσαν να συνδεθούν με το Internet μέσω αισθητήρων. Ο Ashton επινόησε τον όρο για να τονίσει τη δύναμη των συστημάτων ταυτοποίησης μέσω ραδιοσυχνοτήτων, γνωστά ως RFID (Radio Frequency Identification) [12], που χρησιμοποιούνται από εταιρικές εφοδιαστικές αλυσίδες, προκειμένου να μετρήσουν και να παρακολουθούν τα εμπορεύματα χωρίς την ανάγκη για ανθρώπινη παρέμβαση. Σήμερα, το Διαδίκτυο των πραγμάτων έχει γίνει ένας δημοφιλής όρος για την περιγραφή σεναρίων στα οποία η σύνδεση στο Internet και η δυνατότητες των υπολογιστών επεκτείνονται σε μια ποικιλία από αντικείμενα, συσκευές, αισθητήρες και είδη καθημερινής χρήσης.

Ενώ ο όρος «Διαδίκτυο των πραγμάτων» είναι σχετικά καινούριος, η έννοια του συνδυασμού υπολογιστών και τα δικτύων για παρακολούθηση και έλεγχο συσκευών υπάρχει εδώ και δεκαετίες. Από τα τέλη της δεκαετίας του 1970, για παράδειγμα, τα συστήματα για την εξ αποστάσεως παρακολούθηση μετρητών του ηλεκτρικού δικτύου μέσω τηλεφωνικών γραμμών ήταν ήδη σε εμπορική χρήση. Στη δεκαετία του 1990, οι εξελίξεις στην ασύρματη τεχνολογία επέτρεψε τις "machine-to-machine" (M2M) επιχειρήσεις και τις βιομηχανικές λύσεις που δημιουργήθηκαν για τον έλεγχο και τη λειτουργία των εξοπλισμών , να γίνουν ευρέως διαδεδομένες [13]. Πολλές από αυτές τις αρχικές λύσεις M2M, όμως, βασίστηκαν σε κλειστά δίκτυα με αποκλειστικό σκοπό ή δίκτυα βασισμένα σε ειδικά πρότυπα για βιομηχανίες, αντί για το πρωτόκολλο TCP/IP, που είναι βασισμένο στα πρότυπα του Διαδικτύου.

Χρησιμοποιώντας TCP/IP για να συνδεθούν συσκευές πλην των υπολογιστών στο Διαδίκτυο δεν είναι μια νέα ιδέα. Η πρώτη Internet "συσκευή", μια TCP/IP enabled τοστιέρα που θα μπορούσε να ενεργοποιηθεί και να απενεργοποιηθεί μέσω του Διαδικτύου, παρουσιάστηκε σε ένα συνέδριο για το Διαδίκτυο το 1990. Στα επόμενα χρόνια και άλλα "πράγματα" έγιναν “TCP/IP enabled”, συμπεριλαμβανομένου ενός μηχανήματος αναψυκτικών στο Πανεπιστήμιο Carnegie Mellon των ΗΠΑ και μία μηχανή καφέ στο Πανεπιστήμιο του Cambridge στο Ηνωμένο Βασίλειο (η οποία παρέμεινε

συνδεδεμένη στο Διαδίκτυο μέχρι το 2001). Αυτές οι συσκευές έκαναν την αρχή και έτσι δημιουργηθούν τα θεμέλια για το Διαδίκτυο των πραγμάτων όπως το γνωρίζουμε σήμερα.

3.1 IoT και ασφαλής επικοινωνία στο Διαδίκτυο

Το Διαδίκτυο αποτελεί, στις μέρες μας, αδιαμφισβήτητα ένα καθημερινό και βασικό εργαλείο στη ζωή του σύγχρονου ανθρώπου. Κυριότερες δραστηριότητες, που πραγματοποιούνται δια μέσου αυτού είναι η αναζήτηση πληροφορίας και γνώσης, η επικοινωνία μεταξύ των ατόμων, οι αγορές μέσω διαδικτύου, αλλά και διάφοροι τρόποι ψυχαγωγίας. Γίνεται, συνεπώς, αντιληπτό ότι οι δυνατότητες που προσφέρει το Διαδίκτυο είναι τεράστιες. Αυτό το καθιστά ένα μέσο αρκετά ελκυστικό, διότι με χαμηλό κόστος είναι δυνατό κανείς να εμπλακεί σε πλήθος δραστηριοτήτων.

Η μετάδοση πληροφοριών μέσω του διαδικτύου γίνεται χρησιμοποιώντας τα πρωτόκολλα TCP (Transfer Control Protocol) και IP (Internet Protocol), τα οποία συνιστούν μαζί ένα πρότυπο και καλύπτουν μέρος του ιδεατού μοντέλου για το δίκτυο και ονομάζεται μοντέλο αναφοράς Ανοικτής Διασύνδεσης Συστημάτων, ή μοντέλο αναφοράς OSI. Το Ip πρωτόκολλο καλύπτει όλες τις λειτουργίες, τις οποίες το OSI περιγράφει ως τρίτο επίπεδο ή επίπεδο δικτύου, ενώ το TCP περιλαμβάνει όλες τις λειτουργίες, που συνοψίζονται ως τέταρτο επίπεδο ή επίπεδο Μεταφοράς.

Το Διαδίκτυο των πραγμάτων (Internet of things) συνδέει μέσω διαδικτύου πληθώρα συσκευών, οι οποίες ανταλλάσσουν συνεχώς πληροφορίες. Οι συσκευές ενσωματώνουν ηλεκτρονικά μέσα, λογισμικό, αισθητήρες και διάφορους τρόπους συνδεσιμότητας σε δίκτυο (Wi-Fi, Bluetooth κ.τ.λ.) και μέσω αυτών, γίνεται η συνεχής ανταλλαγή δεδομένων.

Καθώς τα IoT συνεχώς εξαπλώνονται και χρησιμοποιούνται σχεδόν παντού, οι πληροφορίες που ανταλλάσσουν είναι σε πολύ μεγάλο βαθμό σημαντικές έως και απόρρητες. Η θέση ενός αυτοκινήτου σε μια εφαρμογή για parking σε Δήμους, ιατρικά δεδομένα που μεταφέρονται μέσω ενός smartwatch σε μια κεντρική εφαρμογή για επεξεργασία, είναι μερικά παραδείγματα που απαιτούν ασφάλεια στην επικοινωνία των IoT.

3.1.1 Τα πρωτόκολλα SSL και TLS (Transport Layer Security)

Το TLS (Transport Layer Security) πρωτόκολλο, καθώς και ο προκάτοχος αυτού το πρωτόκολλο SSL (Secure Sockets Layer) λειτουργούν πριν το TCP/IP και μετά τις εφαρμογές υψηλού επιπέδου, όπως

είναι για παράδειγμα το HTTP (HyperText Transfer Protocol), το File Transfer Protocol (FTP), το Internet Message Access Protocol (IMAP) κ.α.. Δηλαδή, ανάμεσα στο 4ο ή επίπεδο μεταφοράς και στο 7ο ή επίπεδο εφαρμογών του μοντέλου αναφοράς OSI.

Ο βασικός ρόλος αρχικά του SSL πρωτοκόλλου είναι η λήψη πληροφοριών από τις εφαρμογές υψηλότερων επιπέδων, ώστε αυτές να κρυπτογραφηθούν και στη συνέχεια, η μετάδοσή τους στο διαδίκτυο προς τον ηλεκτρονικό υπολογιστή, που έθεσε το αίτημα. Επιπροσθέτως, το TLS εγγυάται ότι κατά την επικοινωνία εξυπηρετητή - πελάτη (server -client) μέσω του διαδικτύου δεν πρόκειται να μεσολαβήσει κάποιος άλλος χρήστης με σκοπό να υποκλέψει το περιεχόμενο της επικοινωνίας.

Ο προκάτοχος του TLS, το SSL, αναπτύχθηκε από την εταιρεία Netscape και σχεδιάστηκε για να παρέχει ασφάλεια κατά την μετάδοση ευαίσθητων ή προσωπικών δεδομένων στο διαδίκτυο. Το SSL χρησιμοποιεί μεθόδους κρυπτογράφησης των δεδομένων, που ανταλλάσσονται μεταξύ δύο συσκευών εγκαθιδρύοντας μία ασφαλή σύνδεση μεταξύ τους μέσω του διαδικτύου. Χρησιμοποιεί το TCP/IP πρωτόκολλο για τη μεταφορά των δεδομένων και είναι ανεξάρτητο από την εφαρμογή, που χρησιμοποιεί ο τελικός χρήστης. Έτσι, μπορεί να παρέχει υπηρεσίες ασφαλούς μετάδοσης πληροφοριών σε πρωτόκολλα ανώτερου επιπέδου, όπως για παράδειγμα το HTTP, το FTP, το telnet και άλλα. Προσφέρει τις διαδικασίες πιστοποίησης του server από τον client, πιστοποίησης του client από τον server και την εγκαθίδρυση ασφαλούς κρυπτογραφημένης επικοινωνίας μεταξύ των δύο.

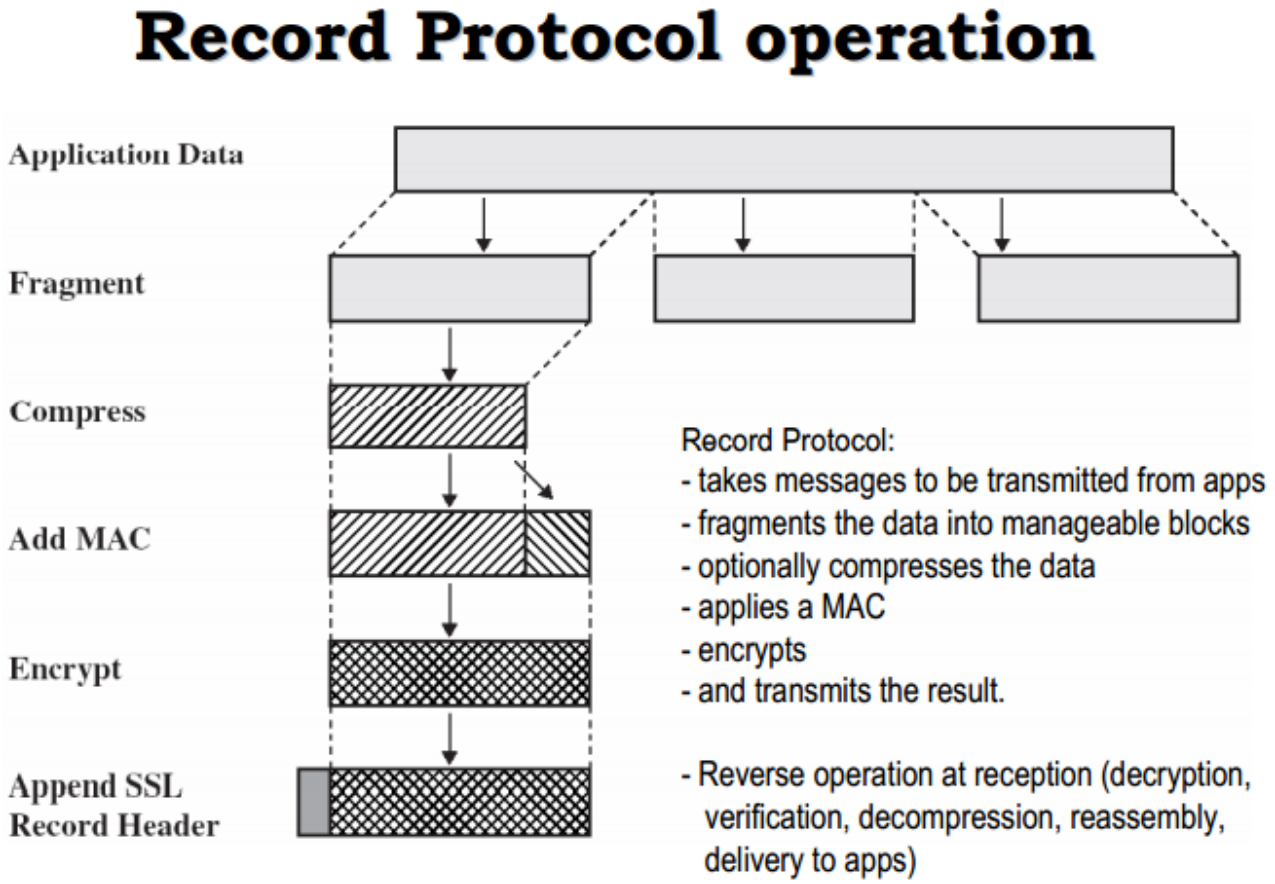
Το SSL βασίζεται σε κρυπτογραφικές τεχνικές Δημόσιου Κλειδιού και οι κρυπτογραφικοί αλγόριθμοι, που υποστηρίζονται από το πρωτόκολλο, είναι οι DES, DSA , KEA, MD5 , RC2/RC4, RSA, SHA-1 (κρυπτογραφική συνάρτηση κατακερματισμού(, SKIPJACK και Triple-DES (κανείς τους δεν αποτελεί ασφαλή αλγόριθμο σήμερα, αλλά ήταν οι επιλογές κατά τις δεκαετίες χρήσης του SSL – το οποίο άλλωστε σήμερα δεν είναι ασφαλές). Υπάρχουν τρεις εκδόσεις SSL (1.0, 2.0 και 3.0): Η έκδοση 1.0 δεν εκδόθηκε δημόσια. Η έκδοση 2.0 εκδόθηκε το Φεβρουάριο του 1995, όμως, δεν είχε ιδιαίτερα μεγάλη απήχηση, διότι περιείχε σημαντικά σφάλματα, λάθη και παραλείψεις και δεν το καθιστούσαν ιδιαίτερα ασφαλές. Το SSL 3.0 εκδόθηκε το 1996 και ήταν ένας επανασχεδιασμός του πρωτοκόλλου, που σχεδιάστηκε από τον Paul Kocher, ο οποίος εργάστηκε με τους μηχανικούς της Netscape Phil Karlton και Alan Freier. Σήμερα, η 3.0 έκδοση του SSL δε θεωρείται πλέον αρκετά αξιόπιστη και ασφαλής: το τελικό «πλήγμα» στην ασφάλειά της αποτέλεσε το 2014 η επίθεση Padding Oracle On Domgraded Legacy Encryption (POODLE).

Έτσι το SSL, έπρεπε να αντικατασταθεί από ένα ισχυρότερο πρωτόκολλο, το οποίο όχι μόνο να δίνει στους χρήστες την αίσθηση της ασφάλειας, αλλά και να παρεμποδίζει όλα τα είδη των παρεμβάσεων

και επιθέσεων. Ο απόγονος του SSL , το TLS εξελίχθηκε σημαντικά και ικανοποιεί τις ανάγκες των χρηστών.

Το TLS περιλαμβάνει δυο στρώματα, το TLS Record Protocol και το TLS Handshake Protocol .

Το TLS Record πρωτόκολλο δομείται σε στρώματα. Σε κάθε στρώμα, τα μηνύματα συμπεριλαμβάνουν πεδία για το μήκος ,την περιγραφή και το περιεχόμενο. Το πρωτόκολλο τμηματοποιεί τα δεδομένα σε μπλοκ, τα συμπιέζει, εφαρμόζει το MAC και κρυπτογραφεί και μεταδίδει το αποτέλεσμα όπως φαίνεται στην παρακάτω εικόνα

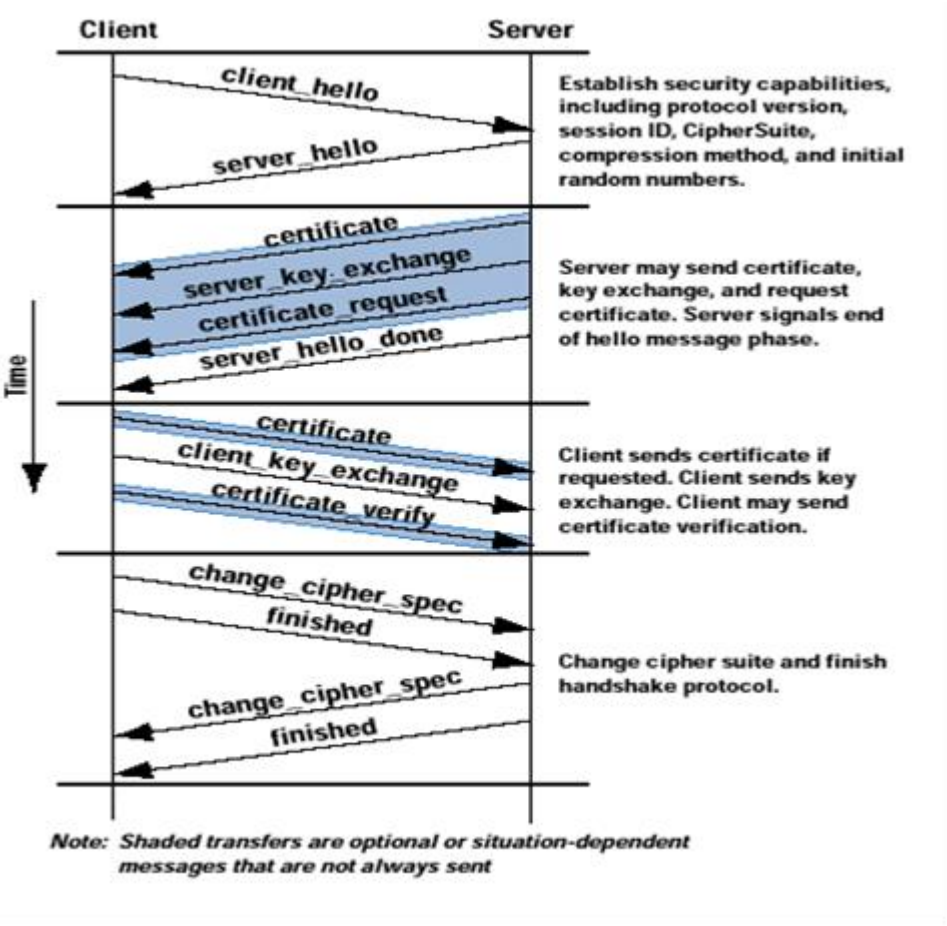


Εικόνα 14. Λειτουργία του TLS Record πρωτοκόλλου

Οι αλγόριθμοι που θα χρησιμοποιηθούν σε κάθε βήμα της λειτουργίας του TLS Record πρωτοκόλλου, συμφωνούνται μεταξύ εξυπηρετητή και πελάτη κατά τη διαδικασία του TLS Handshake πρωτοκόλλου

Το TLS Handshake πρωτόκολλο είναι υπεύθυνο για την αυθεντικοποίηση και την ανταλλαγή κλειδιών αναγκαία για την εγκαθίδρυση ή την επανέναρξη ασφαλών συνόδων. Όταν ένας TLS πελάτης και ο εξυπηρετητής αρχίζουν να επικοινωνούν, συμφωνούν σε μία έκδοση του πρωτοκόλλου, επιλέγουν αλγόριθμο κρυπτογράφησης, σε κάποιες περιπτώσεις αυθεντικοποιούν ο ένας τον άλλον

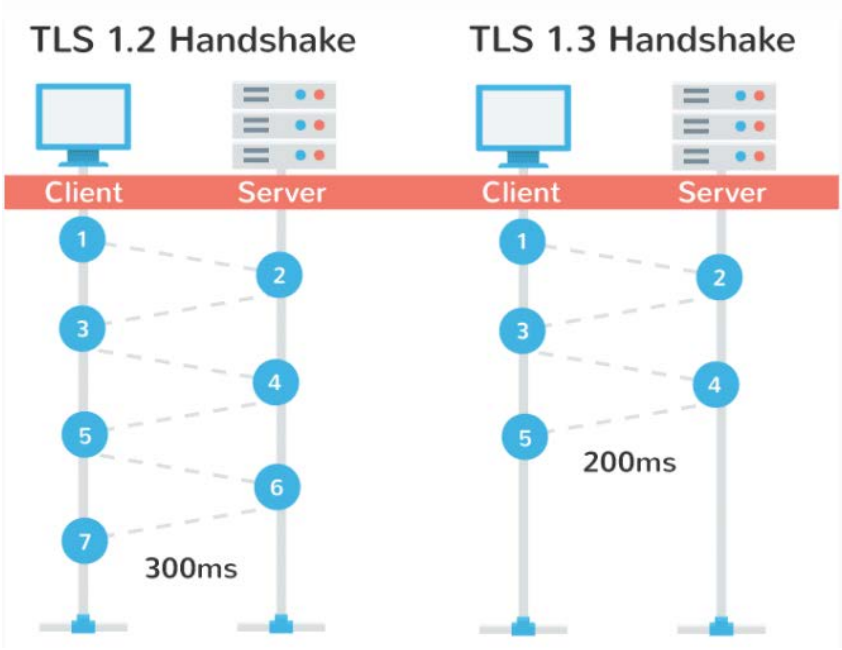
και χρησιμοποιούν τεχνικές δημόσιου κλειδιού κρυπτογράφησης για να παράγουν τα διαμοιραζόμενα μυστικά. Η διαδικασία φαίνεται σχηματικά στην παρακάτω εικόνα.



Εικόνα 15. Το TLS Handshake

Η ανταλλαγή κλειδιών και η αυθεντικοποίηση γίνεται με χρήση των αλγορίθμων, RSA, RSA με χρήση DHE (Diffie-Hellman-Exchange), RSA με χρήση ECDH, δηλαδή Diffie-Hellman με χρήση αλγόριθμων ελλειπτικών καμπυλών και άλλους συνδυασμούς. Η κρυπτογράφηση των δεδομένων γίνεται με AES σε λειτουργία GCM και άλλους συνδυασμούς. Στις εκδόσεις **TLS 1.0 και 1.1** χρησιμοποιούνται οι αλγόριθμοι Hash Message Authentication Code (HMAC)-MD5, HMAC-SHA1, GOST 28147-89 IMIT, GOST R 34.11-94. Ο GOST αλγόριθμος είναι ένας σοβιετικός αλγόριθμος, που βασίζεται σε μία συνάρτηση κατακερματισμού γνωστή με το όνομα GOST Hash Function. Στην έκδοση του **TLS 1.2** εκτός από τους ανωτέρω αλγορίθμους χρησιμοποιούνται οι αλγόριθμοι HMAC-SHA 256/384 και AEAD.

Από τον Αύγουστο του 2018, υπάρχει διαθέσιμο και το **TLS 1.3** [14]**.** Υπόσχεται να κάνει τις επικοινωνίες μας πιο ασφαλής και πιο γρήγορες. Χρησιμοποιεί ένα γύρο για το Handshake, ενώ το TLS 1.2 χρησιμοποιεί δύο. Η παρακάτω εικόνα δίνει σχηματικά τη διαφορά στη διαδικασία.



Εικόνα 16. TLS 1.3 βελτιωμένο handshake

Άλλο ένα νέο χαρακτηριστικό του TLS 1.3 είναι ότι ….. «θυμάται»! Αν έχει ήδη επισκεφθεί κάποιο συγκεκριμένο site μπορεί να αρχίσει να στέλνει κατευθείαν μηνύματα στον server. Αυτό ονομάζεται «μηδενικός γύρος» (zero round trip) για το Handshake. Τα παραπάνω χαρακτηριστικά αυξάνουν την ταχύτητα του πρωτοκόλλου.

Το TLS 1.3 τέλος, δεν υποστηρίζει τους αλγόριθμους που έχουν ευπάθειες και υπήρχαν στο TLS 1.2 για λόγους συμβατότητας. Χαρακτηριστικά αναφέρουμε τους SHA-1, RC4, DES, 3DES, AES-CBC και MD5. Άρα γενικά το πρωτόκολλο είναι πιο ασφαλές και αποκλείει συνδέσεις με ευπαθείς αλγόριθμους.

Γενικά το TLS βασίζεται σε ανοιχτές βάσεις επικοινωνίας, που το καθιστούν πολύ πιο επεκτάσιμο και πιο πιθανό να υποστηρίζει μελλοντική τεχνολογία και μελλοντικούς αλγόριθμους όπως π.χ. τον SATURNIN. Είναι «προς τα πίσω» (backwards) συμβατό, που σημαίνει ότι χρησιμοποιείται ακόμα για πελάτες που υποστηρίζουν το SSL Επιτρέπει ασφαλείς και μη ασφαλείς συνδέσεις πάνω σε συγκεκριμένη θύρα ενώ το SSL επιτρέπει ασφαλείς συνδέσεις μόνο. Δεν απαιτεί συμβολή ή υποστήριξη από κάποιο συγκεκριμένο λειτουργικό σύστημα. Είναι εύκολο στη χρήση και η πιο συχνά χρησιμοποιούμενη ασφάλεια στο διαδίκτυο.

Ως βασικότερο μειονέκτημα, το TLS έχει αυξημένο επεξεργαστικό φόρτο. Αυτό έχει μεγάλη επίδραση στην αποδοτικότητα των IoT, όπου η επεξεργαστική ισχύ είναι περιορισμένη. Προς τούτο σημειώνεται ότι η λεγόμενη «lightweight» κρυπτογράφηση, η οποία περιγράφεται στη συνέχεια, μπορεί να δώσει λύση σε αυτό το πρόβλημα, ως προς το τμήμα του πρωτοκόλλου που αφορά τη συμμετρική κρυπτογράφηση.

Στην περίπτωση των Internet of Things και της παρούσας διπλωματικής εργασίας, θεωρούμε ότι χρησιμοποιείται το πρωτόκολλο TLS με υποστήριξη, από πλευράς συμμετρικών αλγορίθμων, του AES και του SATURNIN.

3.2 IoT και κρυπτανάλυση

Η παροχή ασφάλειας και ιδιωτικότητας στο IoT εξακολουθεί να είναι μια πρόκληση, λόγω του τεράστιου αριθμού ετερογενών συσκευών, αλλά και στην ανταλλαγή δεδομένων μέσω ανασφαλών συνδέσεων. Επίσης αντιμετωπίζουμε και θέματα δικτύου (αυθεντικοποίηση συσκευών, πρόσβαση). Για παράδειγμα, η δημιουργία ψεύτικων δικτύων (fake networks-termed botnets) από επιτιθέμενους, για υποκλοπή δεδομένων και στοιχείων ιδιωτικότητας [15], αποτελεί έναν υπαρκτό κίνδυνο.

Στα IoT, αρκετές τεχνολογίες έχουν αναπτυχθεί για να προσφέρουν ασφάλεια και ιδιωτικότητα των πληροφοριών, όπως το TLS που αναφέραμε παραπάνω, που βελτιώνει την εμπιστευτικότητα και την ακεραιότητα. Επίσης η δρομολόγηση πολλαπλών στρωμάτων (onion routing), που κρυπτογραφεί και αναμειγνύει την κυκλοφορία του διαδικτύου από διαφορετικές πηγές, και κρυπτογραφεί τα δεδομένα σε πολλαπλά επίπεδα-στρώματα (layers) χρησιμοποιώντας δημόσια κλειδιά στη διαδρομή μετάδοσης τους [16].

Η ασφάλεια ενός κρυπτογραφικού συστήματος έγκειται στη μυστικότητα του κλειδιού του. Επομένως κάποιος επιτιθέμενος που θέλει να ανακτήσει από το κρυπτοκείμενο το αρχικό κείμενο, χωρίς φυσικά να ξέρει το κλειδί κρυπτογράφησης, θεωρούμε ότι γνωρίζει αναλυτικά τον αλγόριθμο κρυπτογράφησης. Έχουμε τις παρακάτω κατηγορίες κρυπτανάλυσης (όχι μόνο σε κρυπτογραφικούς αλγορίθμους εντός IoT αλλά σε οποιοδήποτε σενάριο):

- Επιθέσεις κρυπτογραφήματος (ciphertext-only attack)
- Επιθέσεις γνωστού απλού κειμένου (known plaintext attack)
- Επιθέσεις επιλεγμένου απλού κειμένου (chosen plaintext attack)
- Επιθέσεις προσαρμοσμένου επιλεγμένου απλού κειμένου (adaptive chosen plaintext attack)
- Επιθέσεις επιλεγμένου κρυπτογραφήματος (chosen ciphertext attack)
- Επιθέσεις προσαρμοσμένου επιλεγμένου κρυπτογραφήματος (adaptive chosen ciphertext attack)

Κάθε γνωστή τεχνική κρυπτανάλυσης υπάγεται σε κάποια από τις ως άνω κατηγορίες.

Εκτός των επιθέσεων, που αφορούν την αντιστοιχία κειμένου - κρυπτοκειμένου, υπάρχουν και οι κλασικές μέθοδοι επίθεσης που προσπαθούν να «μαντέψουν» το κλειδί ή τα κλειδιά. Οι σημαντικότερες είναι:

- Επιθέσεις εξαντλητικής αναζήτησης (Brute force attacks): Ο επιτιθέμενος μπορεί να δοκιμάσει όλα τα δυνατά κλειδιά αποκρυπτογράφησης μέχρι να βρει το κλειδί που έχει χρησιμοποιηθεί.
- Επιθέσεις επιλεγμένου-γνωστού κλειδιού (chosen-known key attack): Ο επιτιθέμενος δεν μπορεί να επιλέξει το κλειδί κρυπτογράφησης-αποκρυπτογράφησης, αλλά έχει κάποια γνώση για τη σχέση διαφορετικών κλειδιών και χρησιμοποιώντας αυτή τη γνώση προσπαθεί να καθορίσει νέα κλειδιά.
- Επιθέσεις λεξιλογίου (dictionary attacks) : Η επίθεση αυτή συνήθως αφορά τους κωδικούς (passwords), χρησιμοποιώντας κωδικούς μέσα από γνωστές λίστες.
- Επιθέσεις χρονομέτρησης (timing attacks) : Αυτές εκμεταλλεύονται τη συσχέτιση μεταξύ ενός κρυπτογραφικού κλειδιού και τη χρονική διάρκεια μιας κρυπτογραφικής πράξης που χρησιμοποιεί το συγκεκριμένο κλειδί. Επειδή η διάρκεια υπολογισμού μιας πράξης συχνά εξαρτάται από τα δεδομένα εισόδου, π.χ. το πλήθος των 1 στο κλειδί, η χρονομέτρηση αποκαλύπτει κάποια πληροφορία για αυτά.
- Διαφορική ανάλυση σφαλμάτων (differential fault analysis) : Βασίζονται στο γεγονός ότι διάφορα σφάλματα σε κρυπτογραφικές πράξεις που συσχετίζονται με ένα συγκεκριμένο κρυπτογραφικό κλειδί, μπορεί να διαρρεύσουν πληροφορίες για το εν λόγω κλειδί.
- Ανάλυση σφάλματος (fault analysis) : Έχουν κοινά με την προηγούμενη κατηγορία αλλά είναι διαφορετική οικογένεια. Βασίζεται στο γεγονός ότι πολλές υλοποιήσεις κρυπτογραφικών πράξεων επιστρέφουν μηνύματα λάθους σε περίπτωση αποτυχίας.

Ειδικά στα IoT , όλα τα παραπάνω μπορούν να εφαρμοστούν με αρκετά μεγάλη επιτυχία γιατί η μειωμένη επεξεργαστική ισχύ των συσκευών, η περιορισμένη μνήμη RAM αλλά και η πολύ μεγάλη ποικιλία των ολοκληρωμένων που χρησιμοποιούνται οδηγούν συχνά σε διαφορετικές υλοποιήσεις των αλγορίθμων κρυπτογράφησης. Σκοπός των κατασκευαστών είναι να φτιάξουν αλγόριθμους που να μπορούν να λειτουργούν στα περιορισμένα resources της κάθε συσκευής, αφήνοντας έτσι πολλές ευπάθειες στον κώδικα υλοποίησης, που μπορούν να εκμεταλλευθούν οι παραπάνω επιθέσεις.

Κεφάλαιο 4
Κβαντικοί Υπολογιστές

Ο κβαντικός υπολογιστής είναι μία υπολογιστική συσκευή που εκμεταλλεύεται χαρακτηριστικές ιδιότητες της κβαντομηχανικής, όπως είναι η αρχή της υπέρθεσης και της διεμπλοκής καταστάσεων, νια να φέρει εις πέρας επεξεργασία δεδομένων και εκτέλεση υπολογισμών. Ένας κβαντικός υπολογιστής μπορεί να ερευνήσει με πρωτοφανή ταχύτητα τεράστιες και αδόμητες βάσεις δεδομένων, να σπάσει κάθε γνωστό κρυπτογραφικό κώδικα, να προσομοιώσει πολύπλοκες διεργασίες, και να επιλύσει προβλήματα τα οποία είναι αδύνατον να λυθούν από τους κλασικούς ηλεκτρονικούς υπολογιστές οι οποίοι πλέον ονομάζονται «συμβατικοί» .

Η πρώτη πώληση ενός κβαντικού υπολογιστή έγινε στην εταιρία Lockheed Martin που είναι η μεγαλύτερη προμηθεύτρια του αμερικανικού πενταγώνου. Αγόρασε έναν κβαντικό υπολογιστή από την εταιρία D-wave (https://www.dwavesys.com/) . Ο D-wave One , όπως ονομάζεται ο κβαντικός υπολογιστής μοιάζει με ένα τεράστιο μαύρο κουτί. Το μόνο που είναι γνωστό είναι ότι χρησιμοποιείται ένα υπεραγώγιμο τσιπ των 128 qubits που ονομάζεται Rainier. Ο επεξεργαστής αυτός είναι θωρακισμένος με ειδικά φίλτρα για την προστασία του από κάθε είδους εξωτερικό παράγοντα ώστε να μην υπερθερμαίνεται.

Εικόνα 17. Ο D-wave One

Όταν οι κβαντικοί υπολογιστές θα είναι διαθέσιμοι για εμπορική και προσωπική χρήση θα μπορούν να «σπάσουν» την εμπιστευτικότητα, την ιδιωτικότητα και την ακεραιότητα των πληροφοριών με σχετική ευκολία.

4.1 Bits & Qubits

Στον κλασικό ηλεκτρονικό υπολογιστή στοιχειώδης μονάδα πληροφορίας είναι το **bit**, ενώ σε έναν κβαντικό υπολογιστή το **qubit** (κβαντικό δυαδικό ψηφίο). Τα bit μπορούν να αναπαραστήσουν την τιμή 1 ή 0 ενώ το qubit μπορεί να αναπαραστήσει την τιμή 1,0, ή οποιαδήποτε υπέρθεση αυτών των 2. Δύο qubits μπορούν να αναπαραστήσουν οποιαδήποτε υπέρθεση τεσσάρων δυνατών καταστάσεων, 3 qubits οποιαδήποτε υπέρθεση 8 καταστάσεων. Γενικά ένας κβαντικός υπολογιστής με **n qubits** μπορεί να βρίσκεται σε αυθαίρετη υπέρθεση έως **2^n δυνατών καταστάσεων ταυτόχρονα**, ενώ ένας κλασικός υπολογιστής μπορεί να βρίσκετε μόνο σε μια από αυτές τις καταστάσεις κάθε στιγμή.

Έτσι, ένας κβαντικός υπολογιστής , μπορεί να επεξεργαστεί πληροφορία qubit, με τη δυνατότητα να βρίσκεται σε κατάσταση επαλληλίας, με αποτέλεσμα τα qubit να μπορούν να βρίσκονται ταυτόχρονα και "πάνω" και "κάτω" (τιμή 0 και 1 ταυτόχρονα). Κάθε qubit λοιπόν παραμένει σε διάφορες καταστάσεις επαλληλίας, οπότε το υπολογιστικό πρόγραμμα εκτελείται ταυτόχρονα ή παράλληλα και για τις δύο τιμές της δυαδικής μεταβλητής του κάθε qubit. Αυτό το φαινόμενο είναι γνωστό ως μαζικός κβαντικός παραλληλισμός και αποτελεί το θεμελιώδη μηχανισμό λειτουργίας ενός κβαντικού υπολογιστή. Ο παρακάτω πίνακας δίνει ένα παράδειγμα της κατάστασης επαλληλίας.

Qubits	ΚΑΤΑΧΩΡΕΙ ΤΑΥΤΟΧΡΟΝΑ	ΑΝΤΙΣΤΟΙΧΑ bits
1	(0 και 1)	2
2	(0 και 1) (0 και 1)	$2x2=2^2=4$
3	(0 και 1) (0 και 1) (0 και 1)	$2x2x2=2^3=8$
⋮	⋮	⋮
300	(0 και 1) (0 και 1)..... (0 και 1)	2^{300}

Εικόνα 18. Κατάσταση επαλληλίας

Μια πολύ χρήσιμη παράσταση για το qubit, η οποία απεικονίζει σχεδόν όλα τα χαρακτηριστικά του, είναι η σφαίρα Bloch. Είναι μια γεωμετρική αναπαράσταση του χώρου καθαρής κατάστασης ενός κβαντομηχανικού συστήματος δύο επιπέδων που πήρε το όνομά της από τον φυσικό Felix Bloch. Εναλλακτικά, είναι ο χώρος καθαρής κατάστασης ενός κβαντικού καταχωρητή 1 qubit. Με τη σφαίρα Bloch δίνουμε γεωμετρικά την αντιστοιχία μεταξύ των στοιχείων της σφαίρας και των καθαρών καταστάσεων του qubit.

Εικόνα 19. Η σφαίρα Bloch

Άρα, ένας κβαντικός υπολογιστής μπορεί να θεωρηθεί ως ένα σύνολο από n qubits το όποιο λέγεται και «κβαντικός καταχωρητής» (quantum register). Ένας quantum register που περιέχει n qubits μπορεί να έχει 2^n επιτρεπτές καταστάσεις και κάθε κατάσταση σε αυτόν τον υπολογιστή παράγεται από την υπέρθεση (superposition) των καταστάσεων των qubits.

4.2 Μετακβαντική Κρυπτανάλυση

Έστω ότι έχουμε ένα τηλεφωνικό κατάλογο ο οποίος ως γνωστό έχει τους αριθμούς με βάση την αλφαβητική σειρά των ονομάτων. Είναι δομημένος δηλαδή με βάση τα ονόματα οπότε μπορεί κάποιος να βρει εύκολα τον αριθμό αν γνωρίζει το όνομα. Αν όμως κάποιος γνωρίζει τον αριθμό και ψάχνει να βρει το όνομα στο οποίο αντιστοιχεί, τότε αυτό είναι πάρα πολύ δύσκολο. Αυτό το πρόβλημα καλείται να λύσει **ο κβαντικός αλγόριθμος του Grover** [17] και μάλιστα πιο γρήγορα από έναν κλασικό αλγόριθμο.

Για να λύσει αυτό το πρόβλημα ο κλασικός υπολογιστής, θα έπρεπε να έχει έναν καταχωρητή με δύο bit όπου στο ένα θα είχε καταχωρηθεί ο γνωστός αριθμός και στο άλλο θα εισάγονταν κάθε φορά ένας αριθμός από τον κατάλογο με το αντίστοιχο όνομα και εάν οι αριθμοί ήταν ίσοι, τότε θα σταματούσε η αναζήτηση ενώ αν δεν ήταν, θα προχωρούσε στον επόμενο αριθμό του καταλόγου μέχρι να βρει το ζητούμενο. Άρα αν οι αριθμοί του καταλόγου ήταν N, τότε το επιθυμητό αποτέλεσμα της διαδικασίας θα ερχόταν στην καλύτερη περίπτωση μετά από 1 προσπάθεια και στη χειρότερη μετά από N. Ο μέσος όρος δηλαδή N/2 προσπάθειες. Ο κβαντικός αλγόριθμος του Grover υπόσχεται ότι η διαδικασία μπορεί να γίνει με μόνο $\sqrt{N}$ προσπάθειες. Αυτό σημαίνει ότι αν ο κατάλογος περιέχει 1000000 αριθμούς, ο κλασικός υπολογιστής θα πρέπει να κάνει 500000 κατά μέσο όρο προσπάθειες ενώ ο κβαντικός με τον αλγόριθμο του Grover μόνο 1000!

Πρακτικά αυτό σημαίνει ότι αν θέλουμε να βρούμε το κλειδί ενός **συμμετρικού block cipher**, χρησιμοποιώντας τον αλγόριθμο του Grover σε κβαντικό υπολογιστή, είναι σαν να υποδιπλασιάζουμε το μέγεθος του κλειδιού σε έναν «συμβατικό». Π.χ. αν έχουμε τον AES με κλειδί 128bit , χρησιμοποιώντας έναν κβαντικό υπολογιστή και τον αλγόριθμο του Grover, θα είχαμε το ισοδύναμο του να είχαμε τον AES με κλειδί 64bit και έναν συμβατικό υπολογιστή.

Άρα, παρότι θεωρητικά ο αλγόριθμος του Grover μπορεί να επιταχύνει την εύρεση του κλειδιού, αν χρησιμοποιήσουμε κλειδί μεγαλύτερου μήκους, π.χ. 256bit, τότε ο αλγόριθμος ξαναγίνεται ασφαλής. Έτσι, οι συμμετρικοί αλγόριθμοι παραμένουν ασφαλής από κβαντική κρυπτανάλυση. Ο Saturnin , που

θα δούμε αναλυτικά παρακάτω, είναι βασισμένος στον AES και χρησιμοποιεί κλειδί 256bit, συνεπώς αναμένεται να είναι ανθεκτικός και στη μετα-κβαντική εποχή.

Τι γίνεται όμως με τους αλγόριθμους δημοσίου κλειδιού; Υπενθυμίζουμε ότι χρησιμοποιούμε αυτούς τους αλγόριθμους στο TLS πρωτόκολλο για ανταλλαγή του συμμετρικού κλειδιού, έτσι ώστε στην συνέχεια να χρησιμοποιήσουμε το συμμετρικό αλγόριθμο για την επικοινωνία.

Η λογική της μεθόδου με δημόσιο κλειδί είναι ότι είναι εξαιρετικά δύσκολο να γίνει αποκρυπτογράφηση του μηνύματος με γνώση μόνο του δημόσιου κλειδιού (αφού η γνώση του δημόσιου δεν πρέπει να επιτρέπει τον υπολογισμό του ιδιωτικού). Για παράδειγμα η αποκωδικοποίηση μηνύματος το οποίο έχει κωδικοποιηθεί με τη μέθοδο RSA, είναι ένα πρόβλημα που η λύση του ισοδυναμεί με τη λύση ενός άλλου διάσημου προβλήματος, αυτού της παραγοντοποίησης ενός αριθμού σε πρώτους παράγοντες. Έτσι η ασφάλεια της μεθόδου RSA βασίζεται στην πεποίθηση ότι η παραγοντοποίηση ενός πολύ μεγάλου αριθμού, είναι ένα πρόβλημα πολύ δύσκολο να λυθεί με κλασικούς υπολογιστές

Όμως **ο αλγόριθμος του Shor** [18] για παραγοντοποίηση σε κβαντικούς υπολογιστές , υπόσχεται ακριβώς τη γρήγορη λύση αυτού του προβλήματος και άρα θα μπορούσε να χρησιμοποιηθεί για να «σπάσει» τον RSA, τον Diffie–Hellman, τους αλγορίθμους elliptic curve και όλους τους γνωστούς μηχανισμούς δημοσίου κλειδιού. Ένας κβαντικός υπολογιστής υποβιβάζει με τρομερή ευκολία το επίπεδο ασφάλειας του RSA και του elliptic curve στο ισοδύναμο του αλγορίθμου του Καίσαρα.!

Ομοίως και άλλες μέθοδοι κρυπτογράφησης με δημόσιο κλειδί θα μπορούσαν να «σπάσουν» αν ήταν γνωστοί και άλλοι γρήγοροι αλγόριθμοι για τη λύση λογαριθμικών προβλημάτων. Είναι λοιπόν εντυπωσιακό το πώς η μια μεγάλη ανακάλυψη θα μπορούσε να χρησιμοποιηθεί για να ακυρώσει την άλλη!

Κεφάλαιο 5
"Lightweight" Κρυπτογραφία

Όπως αναφέραμε και στην εισαγωγή, η χρησιμοποίηση μικροσυσκευών όπως τα Radio-Frequency Identification (RFID) tags, οι βιομηχανικοί ελεγκτές (industrial controllers), τα δίκτυα αισθητήρων (sensor nodes) και οι έξυπνες κάρτες (smart cards), συνεχώς αυξάνεται με γεωμετρικούς ρυθμούς. Η μεταγωγή από τους επιτραπέζιους υπολογιστές στις μικροσυσκευές, αναδεικνύει νέα προβλήματα και προκλήσεις στο τομέα της ασφάλειας και της ακεραιότητας των δεδομένων. Είναι πολύ δύσκολο να μεταφέρουμε αυτούσιους τους υπάρχοντες κρυπτογραφικούς αλγόριθμους, στις παραπάνω μικροσυσκευές και να διατηρήσουμε τα ίδια επίπεδα ασφάλειας, υπολογιστικής ισχύος και πόρων. Μην ξεχνάμε ότι οι γνωστοί αλγόριθμοι σχεδιάστηκαν για να λειτουργούν βέλτιστα σε desktop υπολογιστές ή servers και αυτό κάνει δύσκολη, αν όχι ακατόρθωτη, την υλοποίησή τους σε υπολογιστές με πολύ περιορισμένους πόρους. Ακόμα και αν τους υλοποιήσουμε, η απόδοσή τους κυρίως από πλευράς ταχύτητας θα είναι αισθητά περιορισμένη.

Η **"Lightweight" Κρυπτογραφία** είναι ο κλάδος της κρυπτογραφίας που ασχολείται με την εύρεση κρυπτογραφικών αλγορίθμων που θα μπορούσαν να ανταποκριθούν στις δυσκολίες και στις απαιτήσεις μιας μικροσυσκευής με περιορισμένους διαθέσιμους πόρους. Τα τελευταία χρόνια, υπάρχει μια τεράστια προσπάθεια από την ακαδημαϊκή κοινότητα για να βρεθούν υλοποιήσεις των υπαρχόντων αλγορίθμων που θα είναι πιο αποδοτικοί σε ταχύτητα χωρίς να καταναλώνουν πολλούς πόρους, Ταυτόχρονα, σχεδιάζονται νέοι αλγόριθμοι και πρωτόκολλα που θα παρέχουν καλύτερα επίπεδα ασφάλειας χρησιμοποιώντας ελάχιστους πόρους και χωρίς να επηρεάζουν σημαντικά την απόδοση του συστήματος. Οι τελευταίοι , θα αποτελέσουν και την "Lightweight" Κρυπτογραφία του μέλλοντος.

Ποιο φάσμα συσκευών μπορεί να καλύψει η "Lightweight" Κρυπτογραφία; Και ποια η σχέση της με την «συμβατική» κρυπτογραφία που όλοι σήμερα γνωρίζουμε.; Όπως είπαμε πιο πριν, η «συμβατική» κρυπτογραφία έχει αλγόριθμους που είναι βελτιστοποιημένοι για τα desktops και για τους servers. Πολλοί σύγχρονοι επεξεργαστές όπως πχ οι Intel [19], έχουν εντολές ειδικά σχεδιασμένες για την κρυπτογραφία, επιταχύνοντας δραματικά την απόδοση των αλγορίθμων που τις χρησιμοποιούν. Στον αντίποδα, οι μικροσυσκευές , δεν διαθέτουν ανάλογες εντολές και η υπολογιστική τους ισχύς είναι περιορισμένη. Άρα λοιπόν, η «συμβατική» κρυπτογραφία θα μπορούσε να συνεχίζει να καλύπτει το

φάσμα του «high end» computing και η "Lightweight" να καλύψει την ανάγκη των «embedded» συστημάτων, όπως φαίνεται και στην παρακάτω εικόνα.

Servers and Desktops	Conventional cryptography
Tablets and Smartphones	
Embedded Systems	Lightweight cryptography
RFID and Sensor Networks	

Εικόνα 20. Φάσμα συσκευών και κρυπτογραφία

Εδώ όμως είναι απαραίτητο να σημειώσουμε ότι παρότι οι «high end» υπολογιστές δεν χρειάζονται τη "Lightweight" κρυπτογραφία για την ασφάλεια των δεδομένων, πρέπει να την υποστηρίζουν και να μπορούν να τρέχουν τους αντίστοιχους αλγόριθμους. Γιατί εφόσον οι «μικροσυσκευές» ανταλλάσσουν δεδομένα με "Lightweight" κρυπτογραφία, θα πρέπει και οι «ισχυροί» υπολογιστές, οι οποίοι συγκεντρώνουν και επεξεργάζονται αυτά τα δεδομένα, να μπορούν να υποστηρίξουν τους αλγόριθμους της "Lightweight" κρυπτογραφίας.

5.1 Κατηγορίες αλγορίθμων

Όπως και στη «συμβατική» κρυπτογραφία, έτσι και στη "Lightweight", υπάρχουν οι ίδιες κατηγορίες αλγορίθμων και χρησιμοποιούνται για τις ίδιες εφαρμογές. Έτσι, κατά τη διάρκεια των τελευταίων ετών, έχουν προταθεί δεκάδες υλοποιήσεις για όλες τις κατηγορίες, οι οποίες προσφέρουν μεγαλύτερη αξιοπιστία και ταχύτητα σε σχέση με τους «συμβατικούς».

Η μεγαλύτερη διαφορά τους από τους «συμβατικούς» είναι η παραδοχή ότι δεν προορίζονται για μια μεγάλη γκάμα εφαρμογών, οπότε έχουν τη δυνατότητα να δημιουργούν συγκεκριμένες δυσκολίες στον επιτιθέμενο. Για παράδειγμα, τα δεδομένα που μπορεί να έχει στη κατοχή του ο επιτιθέμενος, για ένα συγκεκριμένο κλειδί κρυπτογράφησης, σε ένα "Lightweight" αλγόριθμο θα μπορούσαν να είναι λιγότερα, δυσκολεύοντας την κρυπτανάλυση.

Αυτό βέβαια δεν σημαίνει ότι οι "Lightweight" αλγόριθμοι μπορούν να έχουν πιο χαμηλές απαιτήσεις ως προς την κρυπτογραφική τους ισχύ. ισχυροί. Η αρχή υλοποίησής τους στηρίζεται στην ιδέα ότι πρέπει να έχουν υλοποίηση με καλύτερη αναλογία μεταξύ ασφάλειας, απόδοσης και διαθεσιμότητας πόρων. Ας δούμε λοιπόν τις υπάρχουσες προτάσεις για υλοποίηση των "Lightweight" αλγόριθμων ανά κατηγορία.

5.1.1 Lightweight Block Ciphers

Σε αυτή τη κατηγορία, όλες οι προτάσεις προσπαθούν να ξεπεράσουν σε ασφάλεια και αποδοτικότητα τον AES. Στις περισσότερες περιπτώσεις, τον AES-128.

Οι κυριότερες σχεδιαστικές προσεγγίσεις που χρησιμοποιούνται για να αυξηθεί η απόδοση σε σχέση με τους αντίστοιχους συμβατικούς είναι:

- **Μικρότερα «μπλοκ»**: Για να μειωθούν οι απαιτήσεις σε μνήμη RAM μπορούν να χρησιμοποιηθούν μικρότερα μπλοκ. Π.χ. αντί για 128bits που χρησιμοποιεί ο AES, να χρησιμοποιήσουμε 64 ή 80 bits. Βέβαια με αυτό το τρόπο, περιορίζουμε τον μέγιστο αριθμό μπλοκ αρχικού κειμένου που μπορεί να κρυπτογραφηθεί με ασφάλεια. Γενικά ισχύει ότι, όσο μικρότερο το block τόσο ενδέχεται να μειώνεται η ασφάλεια. Έτσι μπορεί τελικά ,να καταλήξουμε στην ανάκτηση του αρχικού κειμένου, ή του κλειδιού. Αν όμως το μήνυμα είναι αρκετά μικρό (όπως συνήθως συμβαίνει στα Internet of Things), η ασφάλεια διατηρείται σε μεγάλο βαθμό.
- **Μικρότερα μεγέθη κλειδιού**: Κάποιοι "Lightweight" αλγόριθμοι χρησιμοποιούν μικρότερα μεγέθη κλειδιού. Π.χ. ο PRESENT χρησιμοποιεί 80-bit. Αυτό όμως δεν είναι αρκετό για την διατήρηση της ασφάλειας των δεδομένων, ιδίως στη μετακβαντική εποχή. Ήδη ο NIST έχει θέσει ως μίνιμουμ μέγεθος κλειδιού τα 112 bits. [20]
- **Πιο «απλοί» γύροι**: Η υλοποιήσεις των συναρτήσεων που ασχολούνται με τους «γύρους» στη "Lightweight" κρυπτογραφία μπορούν να είναι πιο απλές. Π.χ. το S-box θα μπορούσε να είναι 4-bit αντί για 8-bit. Έτσι θα έχουμε αύξηση στην απόδοση αλλά και μείωση της απαίτησης για μνήμη RAM. Στον αντίποδα αναφέρουμε ότι ο αλγόριθμος πρέπει να επαναλαμβάνεται περισσότερες φορές για να διατηρηθεί η ασφάλεια.
- **Απλούστερη διαχείριση κλειδιών**: Η πολυπλοκότητα στη διαχείριση των κλειδιών αυξάνει τη κατανάλωση μνήμης και ενέργειας. Έτσι χρησιμοποιούμε πιο απλή διαδικασία στην διαχείριση κλειδιών, οι οποίες παράγουν υπό-κλειδιά πολύ γρήγορα. Αυτή η προσέγγιση μπορεί να είναι ευάλωτη σε επιθέσεις, ιδίως αν χρησιμοποιηθούν κλειδιά σχετικά μεταξύ τους, ή αδύναμα κλειδιά, ή ακόμη χειρότερα, τα ίδια κλειδιά. Για να αποφύγουμε αυτές τις ευπάθειες μπορούμε να χρησιμοποιήσουμε μια συνάρτηση παραγώγων για την παραγωγή κλειδιών. [21] [22] [23]
- **Απλούστερη υλοποίηση**: Οι «συμβατικοί» αλγόριθμοι έχουν διαφορετικούς τρόπους λειτουργίας και υποστηρίζουν πολλά πρωτόκολλα. Ο περιορισμός λοιπόν των τρόπων λειτουργίας και η υποστήριξη ενός πρωτόκολλου θα καθιστούσε τους "Lightweight" αλγόριθμους πιο απλούς και άρα πιο γρήγορους.

Χαρακτηριστικά παραδείγματα "Lightweight" αλγορίθμων που υποστηρίζουν κάποια από τα παραπάνω χαρακτηριστικά είναι ο DESL [24] ο οποίος είναι μια παραλλαγή του DES, ο PRESENT [25], ο SIMON και ο SPECK [26]. Στα παραπάνω συμπεριλαμβάνεται και η οικογένεια αλγορίθμων SATURNIN για τους οποίους θα μιλήσουμε παρακάτω.

5.1.2 Lightweight Hash Functions

Οι «συμβατικοί» Hash αλγόριθμοι δεν είναι ιδανικοί για μικροσυσκευές κυρίως λόγω του μεγάλου μεγέθους της εσωτερικής τους κατάστασης και τις μεγάλες απαιτήσεις σε υπολογιστική ισχύ. Έτσι υλοποιήθηκαν διάφοροι αλγόριθμοι Hash για "Lightweight" κρυπτογραφία με κυριότερα χαρακτηριστικά τα παρακάτω:

- **Περιορισμένο μέγεθος εσωτερικής κατάστασης**: Μπορεί να βρει εφαρμογή σε περιπτώσεις που δεν απαιτούν μεγάλη αντοχή σε συγκρούσεις (όταν για δυο διαφορετικές εισόδους, η συνάρτηση Hash παράγει ίδια έξοδο), μειώνοντας το μέγεθος της εσωτερικής κατάστασης. Σε περίπτωση βέβαια που η αντοχή σε συγκρούσεις είναι απαραίτητη, τότε ο αλγόριθμος πρέπει να έχει επιπλέον αντοχή προαπεικόνισης (preimage resistance) και αντοχή δεύτερης προαπεικόνισης (second preimage resistance)
- **Μικρότερο μέγεθος μηνύματος**: Οι «συμβατικοί» Hash αλγόριθμοι μπορούν να διαχειρίζονται μεγάλα μηνύματα μεγέθους (γύρω στα 264 bits). Στους "Lightweight", τα μηνύματα είναι πολύ μικρότερα (με ανώτατη τιμή τα 256 bits). Έτσι, οι αλγόριθμοι που βελτιστοποιούνται για μικρά μηνύματα, αυξάνουν την απόδοση.

Παραδείγματα "Lightweight" Hash αλγόριθμων είναι οι PHOTON [27], Quark [28], SPONGENT [29] και ο Lesamnta-LW [30].

5.1.3 Lightweight Message Authentication Codes

Οι αλγόριθμοι message authentication code (MAC) παράγουν μια «ετικέτα» (tag) από το μήνυμα και το κλειδί και χρησιμοποιείται για να προσδιορίσει την αυθεντικότητα και την ακεραιότητα του μηνύματος. Τα «tag» τυπικά έχουν μέγεθος 64 bits. Για μερικές κατηγορίες εφαρμογών όπως π.χ. το VoIP (Voice over IP), το να λάβει ο αποδέκτης ένα μήνυμα με λάθος αυθεντικοποίηση , δεν επηρεάζει τη συνολική ασφάλεια. Άρα εκεί μπορεί να χρησιμοποιηθούν και μικρότερα μεγέθη στα «tag». Υλοποίηση αυτής της ιδέας συναντάμε στους "Lightweight" MAC αλγόριθμους Chaskey [31], TuLP [32] και LightMAC [33].

5.1.4 Lightweight Stream Ciphers

Το 2007, η Ευρωπαϊκή Ένωση και το European Network of Excellence for Cryptology, οργάνωσε ένα διαγωνισμό για την υλοποίηση νέων Lightweight Stream Ciphers, οι οποίοι θα μπορούσαν να χρησιμοποιηθούν καθολικά σε όλο το φάσμα των συσκευών. Έγινε γνωστό ως eSTREAM project [34].

Οι φιναλίστ του παραπάνω διαγωνισμού, οι οποίοι έχουν εξαιρετικές επιδόσεις σε μικροσυσκευές, εφόσον είναι υλοποιημένοι σε hardware, είναι οι:

- **Grain**: Έχει αναλυθεί πάρα πολύ και θεωρείται ασφαλής. Μπορεί να υλοποιηθεί με πολλούς τρόπους, ανάλογα την πλατφόρμα. Επίσης μπορεί να κάνει αυθεντικοποίηση [35]. Η παραλλαγή του Grain και συγκεκριμένα ο Grain-128AEAD έγινε αποδεκτός και από τον NIST ως υποψήφιος Lightweight αλγόριθμος. [36]
- **Trivium**: Επίσης έχει αναλυθεί σε βάθος. Το μόνο μειονέκτημα είναι ότι υποστηρίζει κλειδιά 80-bit [37].
- **Mickey**: Δεν έχει αναλυθεί αρκετά, σε σχέση με τους προηγούμενους. Δεν έχει ευελιξία ως προς την υλοποίηση και υπάρχουν ενδείξεις ότι έχει ευπάθειες [38].

5.2 Ο διαγωνισμός του NIST

Το National Institute of Standards and Technology (NIST), διεξάγει ένα δημόσιο διαγωνισμό για την αποδοχή ενός ή περισσοτέρων Lightweight αλγορίθμων με σκοπό τη χρησιμοποίησή τους σε περιβάλλοντα με περιορισμένη επεξεργαστική ισχύ. Η πρώτη επίσημη προκήρυξη του διαγωνισμού έγινε το Μάιο του 2018 [39].

Μέχρι τον Φεβρουάριο του 2019, 57 υποψήφιοι αλγόριθμοι είχαν υποβληθεί για αξιολόγηση. Από αυτούς, τον Απρίλιο του 2019, οι 56 έγιναν δεκτοί για τον «πρώτο γύρο» της αξιολόγησης. Έτσι ολοκληρώθηκε η πρώτη φάση της «NIST Lightweight Cryptography Standardization Process».

Παρ' όλα αυτά, ο αριθμός των υποψήφιων αλγορίθμων παρέμενε μεγάλος και έτσι αποφασίστηκε να απορριφθούν οι αλγόριθμοι που τα χαρακτηριστικά τους δεν ήταν ελπιδοφόρα και να επικεντρωθεί η ανάλυση μόνο σε αυτούς που υπόσχονταν μέγιστη δυνατή ασφάλεια και απόδοση. Συνολικά 32 υποψήφιοι αλγόριθμοι κατάφεραν να περάσουν στην επόμενη φάση.

Τον Αύγουστο του 2019, ο NIST ανακοίνωσε του αλγόριθμους οι οποίοι συνεχίζουν στην δεύτερη φάση [40]. Αυτοί φαίνονται συνοπτικά, κατά αλφαβητική σειρά, στον παρακάτω πίνακα:

ACE	ISAP	SKINNY-AEAD &
ASCON	KNOT	SKINNY-HASH
COMET	LOTUS-AEAD & LOCUS-AEAD	SPARKLE (SCHWAEMM
DryGASCON	mixFeed	and ESCH)
Elephant	ORANGE	SPIX
ESTATE	Oribatida	SpoC
ForkAE	PHOTON-Beetle	Spook
GIFT-COFB	Pyjamask	Subterranean 2.0
Gimli	Romulus	SUNDAE-GIFT
Grain-128AEAD	SAEAES	TinyJAMBU
HyENA	Saturnin	WAGE
		Xoodyak

Για τους παραπάνω αλγόριθμους το NIST προέτρεψε τους δημιουργούς να προβούν σε βελτιώσεις ή και σε παραλλαγές αν θεωρούν ότι χρειάζεται, με καταληκτική ημερομηνία αλλαγών, το Σεπτέμβριο του 2019. Μετά, καμμιά αλλαγή δεν έγινε δεκτή, και ο NIST προχωρά έκτοτε στην ανάλυση και αξιολόγηση των παραπάνω αλγορίθμων. Όπως γίνεται και σε κάθε διαγωνισμό του NIST, όλη η επιστημονική και ακαδημαϊκή κοινότητα εξετάζει τους ως άνω αλγορίθμους.

Στους αλγορίθμους που έγιναν δεκτοί στην τελική φάση , είναι και η οικογένεια αλγορίθμων **Saturnin** με την οποία ασχολούμαστε στην παρούσα διατριβή και θα παρουσιάσουμε αναλυτικά στο επόμενο κεφάλαιο. Θα μετρήσουμε επίσης την απόδοσή των αλγορίθμων Saturnin σε περιβάλλον μειωμένης επεξεργαστικής ισχύος.

Κεφάλαιο 6
Ο αλγόριθμος SATURNIN

Ο αλγόριθμος Saturnin προτάθηκε στις 29 Μαρτίου 2019 από το National Institute for Research in Digital Science and Technology της Γαλλίας (γνωστό ως INRIA), από το UCL Crypto Group του Βελγίου και από το NCC Group του Καναδά. Συγγραφείς του είναι οι Anne Canteaut, Sebastien Duval, Gaetan Leurent, Maria Naya-Plasencia, Leo Perrin, Thomas Pornin και ο Andre Schrottenloher. [41]

Είναι ένας 256-bit block cipher με κλειδί 256-bit και μία έξτρα παράμετρο μεγέθους 9-bit. Από την θεωρητική περιγραφή του προέκυψαν οι εξής αλγόριθμοι:

- **Saturnin-CTR-Cascade**: Είναι ένας αλγόριθμος αυθεντικοποίησης ο οποίος χρησιμοποίει τον Saturnin σε CTR mode και επιπλέον έναν αλγόριθμο MAC. Η λειτουργία του απαιτεί δύο περάσματα από τα data αλλά δεν χρησιμοποιεί το τμήμα του inverse block cipher.
- **Saturnin-Short**: Προέρχεται από τον Saturnin-CTR και χρησιμοποιείται για μηνύματα μικρού μήκους. Κατά τη διάρκεια της κρυπτογράφησης κάνει μόνο ένα πέρασμα από την κύρια συνάρτηση του Saturnin οπότε διατηρείται η εμπιστευτικότητα και η ακεραιότητα των δεδομένων.
- **Saturnin-Hash**: Είναι μία hash συνάρτηση 256-bit.

Ο αλγόριθμος Saturnin είναι μια πρόταση υλοποίησης για τον αλγόριθμο που προκήρυξε ο οργανισμός NIST στην κατηγορία των lightweight αλγορίθμων οι οποίο όμως να είναι ικανοί να προσφέρουν και μετακβαντική ασφάλεια [42].

Έτσι το εσωτερικό μπλοκ των 128 bits που χρησιμοποιεί ο AES πχ, δεν είναι αρκετό για μετακβαντική ασφάλεια, ακόμα και αν το μέγεθος του κλειδιού είναι 256-bit. Επίσης, στην περίπτωση των Hash συναρτήσεων, προκειμένου να είναι ανθεκτικές (δηλαδή να μην είναι εφικτό να βρίσκονται «συγκρούσεις») στη μετα-κβαντική εποχή, θα πρέπει το μέγεθος της εξόδου μίας τέτοιας συνάρτησης να είναι τουλάχιστον ίσο με 256 bits. Ως εκ τούτου, ο Saturnin Hash χρησιμοποιεί εσωτερικό μπλοκ 256-bit.

Η εργασία των Marc Kaplan, Gaetan Leurent, Anthony Leverrier και Maria Naya-Plasencia με τίτλο « Quantum Differential and Linear Cryptanalysis» [43], ανέδειξε ότι οι υπάρχοντες αλγόριθμοι

υποφέρουν από πολυωνυμικές χρονικές επιθέσεις. Πολλοί ανάγουν αυτή τη δυνατότητα επίθεσης στην επίδραση του «nonce» σε κάθε κλήση του block cipher. Ο Saturnin ισχυρίζεται ότι μπορεί να αντιμετωπίσει αυτή την ευπάθεια.

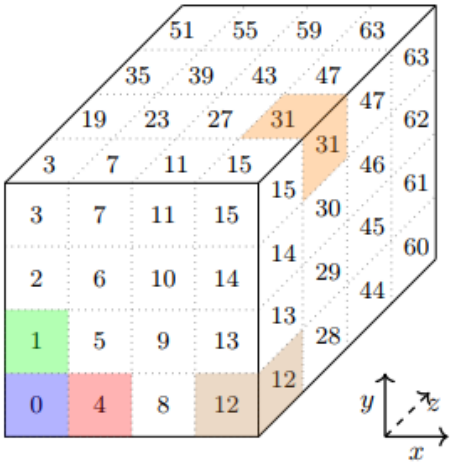
Τέλος, ο αλγόριθμος Saturnin είναι «ελαφρύς» και δεν καταναλώνει πολύ επεξεργαστική ισχύ. Αυτό τον κάνει ικανό να τρέχει σε περιβάλλοντα με πολλές απλές διασυνδεδεμένες συσκευές , όπως αυτές που αποτελούν τα Internet of Things.

Για να πετύχει τα παραπάνω ο Saturnin χρησιμοποιεί ως βάση όλη τη γνώση από τον AES και τον βελτιώνει σε σημεία που θεωρείται «ευπαθής». Άλλωστε ο AES είναι ο αλγόριθμος που έχει ερευνηθεί περισσότερο από οποιαδήποτε άλλον και είναι γνωστά τα πλεονεκτήματα και τα μειονεκτήματά του. Για να μπορέσει δε ο Saturnin να είναι ανθεκτικός και στην κβαντική εποχή εισάγει την ιδέα του τρισδιάστατου εσωτερικού πίνακα, συνολικού μεγέθους 256-bit. Θα μπορούσαμε χαριτολογώντας να πούμε ότι ο Saturnin βασίζεται σε έναν 3-D AES. Εξάλλου, η ομοιότητα του SATURNIN με τον πρότυπο αλγόριθμο AES αποτέλεσε και τον κύριο λόγο επιλογής του συγκεκριμένου αλγορίθμου για την περαιτέρω ανάλυσή μας.

6.1 Χαρακτηριστικά

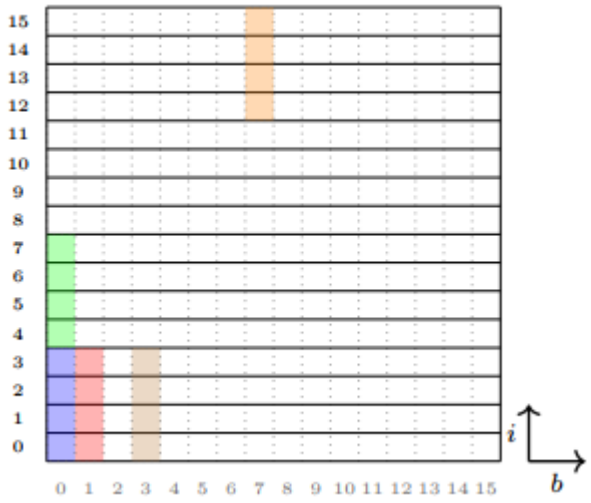
Ο Saturnin χρησιμοποιεί 256-bit μπλοκ, χρησιμοποιώντας 256-bit κλειδί. Επίσης έχει 256-bit εσωτερική δομή.

Τα μπλοκ μπορούν να θεωρηθούν ως κύβος 4 × 4 × 4 με «κομμάτια» (nibbles) των 4-bit όπως φαίνεται παρακάτω:



Εικόνα 21. Nibbles ως κύβοι

Ή, σαν δεκαέξι 16-bit registers, γνωστή και ως "bitsliced παρουσίαση". Η δομή φαίνεται στο παρακάτω σχήμα:



Εικόνα 22. Nibbles ως 16-bit registers

Ή τέλος, ως μια ακολουθία 256 bits.

Στη συνέχεια αυτού του κειμένου θα θεωρήσουμε το μπλοκ ως κύβο από «nibbles».

6.1.1 Εσωτερική δομή και καταστάσεις

Ο Saturnin κάνει διάφορους μετασχηματισμούς που μπορούν να περιγραφούν ως διαφορετικές καταστάσεις στη εσωτερική του δομή. Σύμφωνα με την ορολογία του SHA-3 [44] αυτές είναι:

- **Slice**: Είναι το υποσύνολο των nibbles μέσα στο κύβο όταν η τιμή στον άξονα των z είναι σταθερή.
- **Sheet**: Είναι το υποσύνολο των nibbles μέσα στο κύβο όταν η τιμή στον άξονα των x είναι σταθερή.
- **Columns**: Είναι η τομή του sheet με το slice. Είναι το υποσύνολο των nibbles μέσα στο κύβο όταν η τιμή στον άξονα των x και z είναι σταθερή.

6.1.2 Η λειτουργία του Block Cipher

Ο Saturnin block cipher είναι ένας SPN (Substitution-Permuation Network) αλγόριθμος με άρτιο αριθμό γύρων αρχίζοντας από το 0. Ο αλγόριθμος ορίζει ως υπέρ-γύρο (super-round) τη σύνθεση δύο διαδοχικών γύρων με δείκτες $2r$ and $(2r + 1)$.

Χρησιμοποιεί επίσης μια εσωτερική κατάσταση X των 256-bit και ένα κλειδί K επίσης 256-bit . Και τα δύο επεξεργάζονται ως κύβος (4 × 4 × 4). Τέλος, δύο πρόσθετες λέξεις 16-bit RC0 και RC1 χρησιμοποιούνται για τη δημιουργία των διαδοχικών σταθερών.

Παράμετροι: Ο αλγόριθμος έχει σαν παραμέτρους το R, που είναι ο αριθμός των super-rounds και ανήκει στο διάστημα $\{10, \ldots, 31\}$ με default τιμή το 10, και το D, που είναι ένας 4-bit ακέραιος και χρησιμοποιείται σαν «domain separator».

Αρχικοποίηση: Αρχικά παίρνουν τιμές τα X και K από τα data και από το κλειδί. Και οι δύο ακέραιοι 16-bit RC0 και RC1 αρχικοποιούνται από το παρακάτω 16 bit-string

$$\underbrace{1 \ldots 1}_{7 \text{ ones}} \underbrace{R_4 \ldots R_0}_{R} \underbrace{D_3 \ldots D_0}_{D}$$

όπου R είναι ο αριθμός των super-rounds και D είναι ο «domain separator». Ο γύρος 0 ξεκινά με XOR του K με την εσωτερική κατάσταση (όπως ακριβώς και στον AES).

Λειτουργία των «γύρων» (Round function): Αρχίζοντας από τον «γύρο» 0, ο αλγόριθμος κάνει τη παρακάτω επεξεργασία στην εσωτερική κατάσταση.

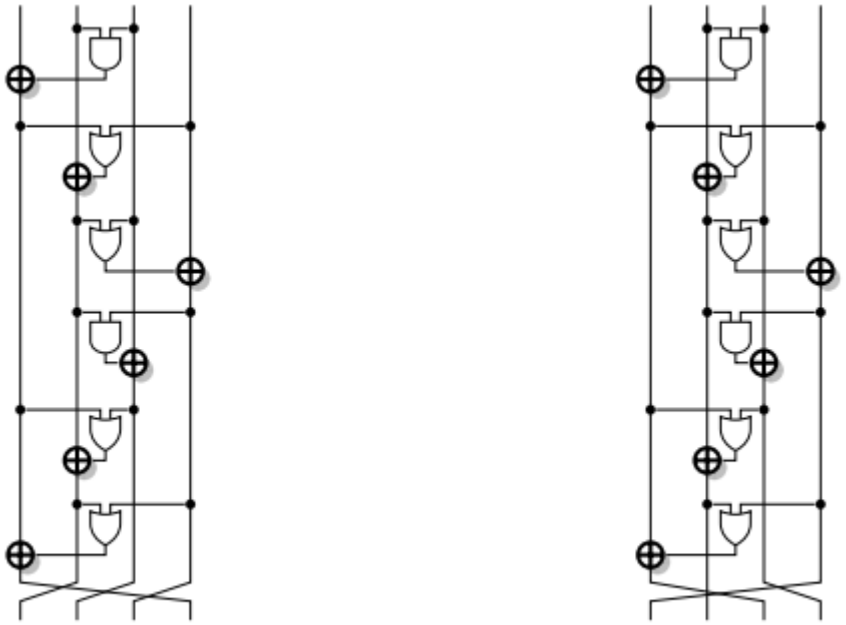
- Ένας S-box layer S, επιδρά με το ίδιο 4-bit Sbox σ_0 σε όλα τα nibbles που έχουν άρτιο αριθμό και το ίδιο 4-bit Sbox σ_1 σε όλα τα nibbles με περιττό αριθμό. Τα Sboxes προκύπτουν από τον παρακάτω πίνακα όπου το x δίνεται σαν

  $$\sum_{i=0}^{3} x_i 2^i = x$$

  και αναφέρεται σε nibble που περιέχει τα $(x3, x2, x1, x0)$.

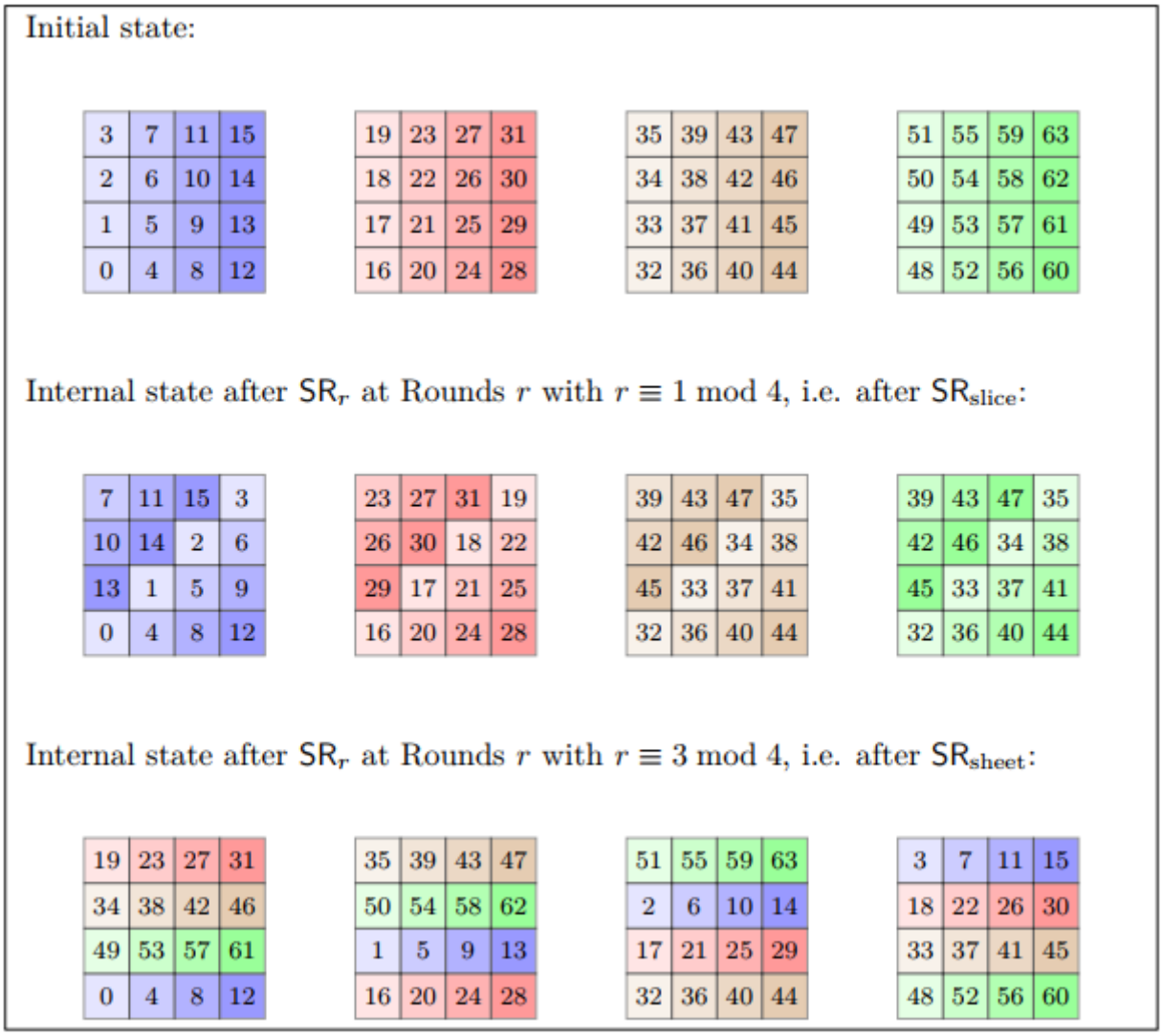
x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_0(x)$	0	6	14	1	15	4	7	13	9	8	12	5	2	10	3	11
$\sigma_1(x)$	0	9	13	2	15	1	11	7	6	4	5	3	8	12	10	14

Εικόνα 23. Ο πίνακας των Sboxes



Εικόνα 24. Σχηματική υλοποίηση των S-boxes

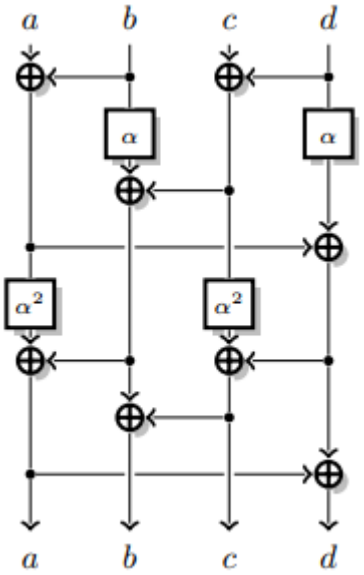
- Ακολουθεί μετάθεση στα nibble και την ονομάζουμε SR_r. Αυτή εξαρτάται από τον τρέχοντα αριθμό «γύρου» r. Μπορούμε να τη δούμε σχηματικά στην παρακάτω εικόνα:



Εικόνα 25. Διαδικασία μετάθεσης

- Μετά έχουμε μία μετατροπή Μ η οποία μπορεί να συγκριθεί με τη συνάρτηση «επόμενου σταδίου» ενός LFSR (Γραμμικός Καταχωρητής Ολίσθησης με Ανάδραση)[1] μήκους 4 σε «Fibonacci mode» και με χαρακτηριστικό πολυώνυμο $X^4 + X^3 + 1$ (το χαρακτηριστικό πολυώνυμο περιγράφει τη συνδεσμολογία του LFSR – περισσότερες λεπτομέρειες εκ φεύγουν του αντικειμένου της παρούσας διατριβής). Η μετατροπή αυτή φαίνεται στην παρακάτω εικόνα.

[1] Ο LFSR είναι ένας μηχανισμός για την παραγωγή του keystream στους αλγόριθμους ροής. Ο καταχωρητής αποτελείται από μία σειρά κελιών (cells) το καθένα από τα οποία αποτελείται από ένα bit. Τα περιεχόμενα των κελιών καθορίζονται από ένα Διάνυσμα Αρχικοποίησης (Initialization Vector) που λειτουργεί σαν το μυστικό κλειδί.



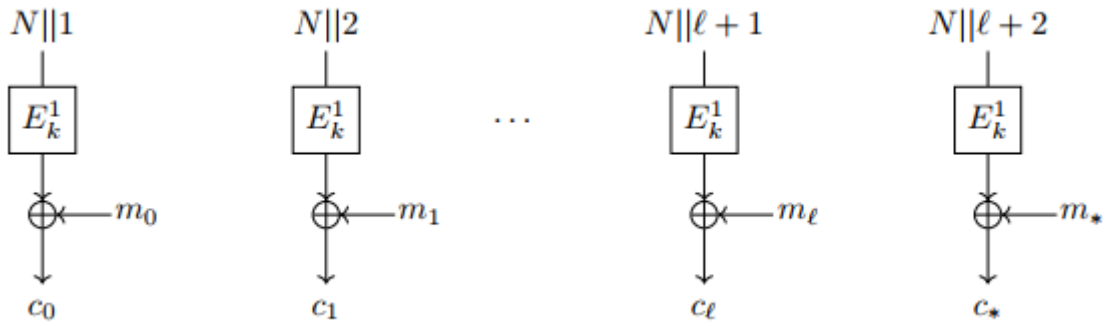
Εικόνα 26. 4 × 4 MDS πίνακας.

Τα(a,b,c,d) αντιστοιχούν στα nibbles (4i, 4i + 1, 4i + 2, 4i + 3), για 0 ≤i ≤ 15

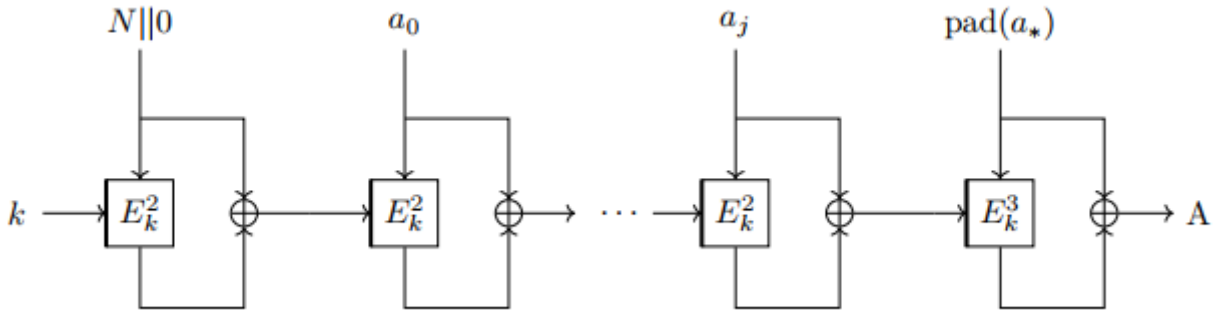
- Συνεχίζει με την αντίστροφη μετάθεση $SR_{r.}^{-1}$.
- Και τέλος, προσθέτει ένα υποκλειδί (subkey) στην περίπτωση που ο αριθμός του «γύρου» είναι περιττός.

6.1.3 Ο Saturnin-CTR-Cascade

Ο Saturnin block cipher σε CTR-Cascade τρόπο λειτουργίας, συνδυάζει τις τεχνικές που είναι ήδη γνωστές για την ανθεκτικότητά τους εναντίον κβαντικών αντιπάλων. Έτσι λοιπόν χρησιμοποιεί τη CTR μέθοδο καθαρά για κρυπτογράφηση και την κατασκευή Cascade για έλεγχο ταυτότητας, ακολουθώντας τη διαδικασία Encrypt-then-MAC. Και οι δύο λειτουργίες φαίνονται σχηματικά παρακάτω:



Εικόνα 27. Counter Mode (CTR) encryption



Εικόνα 28. Cascade construction

Η σταθερά «nonce» έχει μήκος έως 160 bit και η «ετικέτα» έως 256 bit . Το «nonce» γίνεται 161-bit , προσαρτώντας του αρχικά ένα bit με τιμή 1 και έπειτα , όσα bit με τιμή 0 χρειάζονται για να φτάσει σε μέγεθος 161-bit .

6.1.4 Ο Saturnin-Short

Ο Saturnin block cipher σε Short τρόπο λειτουργίας ενδείκνυται για χρήσεις, όπου το μέγεθος των μηνυμάτων είναι πολύ μικρό (προτείνεται μέχρι128 bit) και δεν περιέχουν πρόσθετα δεδομένα. Ο αλγόριθμος εκμεταλλεύεται το γεγονός ότι έχει μέγεθος μπλοκ 256-bit, επιτρέποντας να συνδυάσουμε το «nonce» και το μήνυμα μαζί. Αξίζει να σημειωθεί ότι το κρυπτοκείμενο και η «ετικέτα» δεν είναι δύο ξεχωριστές τιμές όπως στους άλλους τρόπους λειτουργίας. Με τιμή «nonce» N και με ένα κρυπτοκείμενο c, η «ετικέτα» μπορεί να επαληθευτεί με αποκρυπτογράφηση του c και συγκρίνοντας το αριστερό μισό με το N. Ο CTR τρόπος λειτουργίας του φαίνεται παρακάτω.

Εικόνα 29. Ο Saturnin-Short με κλειδί k, μήνυμα m και «nonce» N

6.2 Ασφάλεια του αλγόριθμου

6.2.1 Ενάντια σε «κλασικές» επιθέσεις

Όπως εξηγήσαμε και παραπάνω, η δομή του μπλοκ κρυπτογράφησης του Saturnin είναι πολύ παρόμοια με τη δομή ενός AES που λειτουργεί με λέξεις 16-bit. Έτσι ο block cipher εμπεριέχει την εμπειρία που προκύπτει από την 20ετή κρυπτοαναλυτική προσπάθεια κατά του AES. Άρα, όλες οι οικογένειες επιθέσεων εναντίον του AES μπορούν να μεταφερθούν απευθείας στον Saturnin, αντικαθιστώντας το AES 8-bit Sbox από το Saturnin 16-bit Super-Sbox.

Επειδή έχει πιο απλό key-schedule, καταφέρνει να αυξήσει το μέγιστο αριθμό γύρων για μια επίθεση, από 7 γύρους στον AES, σε 7,5 «υπερ-γύρους» στον Saturnin. Καλύτερες επιθέσεις μπορεί να αυξήσουν αυτόν τον αριθμό έως και σε 8 «υπερ-γύρους». Αυτό όμως είναι εξαιρετικά απίθανο να συμβεί λαμβάνοντας υπόψη τις δυνατότητες των γνωστών εργαλείων κρυπτοανάλυσης.

6.2.2 Ενάντια σε «κβαντικές» επιθέσεις

Σύμφωνα με τους δημιουργούς του Saturnin [41], ένας κβαντικός υπολογιστής, έχοντας στη διάθεσή του μερικά ζεύγη μηνύματος/κρυπτοκειμένου, μπορεί να εκτελέσει μια εξαντλητική αναζήτηση για το κλειδί χρησιμοποιώντας τον αλγόριθμο του Grover [17]. Θα χρειαστεί περίπου $2^{256/2} = 2^{128}$ υπολογισμούς. Σε κάθε περίπτωση θα χρειαστεί πάνω από 2^{112} κρυπτογραφήσεις, καθεμία από τις οποίες περιέχει βασικές λειτουργίες καθώς και αξιολόγηση του Saturnin ως κβαντικού κυκλώματος. Δεν εξετάζουμε τις λεπτομέρειες μιας τέτοιας εφαρμογής, αλλά επισημαίνεται ότι θα απαιτούσε τουλάχιστον τόσο κβαντικές πύλες όσο και κλασικές πύλες (και πιθανώς περισσότερες από τις θεωρητικά αναγκαίες, λόγω της αναγκαιότητας της αναστρεψιμότητας). Στα παραπάνω προσθέτουμε και τα επίπεδα κβαντικής διόρθωσης σφαλμάτων, καθώς οι λειτουργίες διόρθωσης σφαλμάτων είναι κβαντικές μεν, αλλά «κλασικά» ελεγχόμενες και προκαλούν ένα γενικό πρόσθετο κόστος.

Όσον αφορά το super-sbox, ο Saturnin έχει παρόμοιο σχήμα με το AES, με κατάσταση 256 bit. Μέχρι σήμερα, δεν υπάρχει μεγάλη βιβλιογραφία σχετικά με κβαντικές επιθέσεις σε μειωμένους κύκλους AES για την ρύθμιση του μυστικού-κλειδιού, δηλαδή διαδικασίες που ανακτούν το μυστικό κλειδί γρηγορότερα από τον αλγόριθμο του Grover. Μια πρώτη ανάλυση [45] έδειξε ότι οι κβαντικές εκδόσεις της επίθεσης Square μπορούν να στοχεύσουν έως και 6 γύρους AES-128 και 7 γύρους των AES-192 και AES-256 (όπως συμβαίνει και στην κλασική κρυπτανάλυση). Επίσης πραγματοποιούν μια κβαντική

DS-MITM επίθεση σε 8-γύρους AES-256. Μέχρι στιγμής δεν βρέθηκε επίθεση με καλύτερη ταχύτητα. Έτσι, προς το παρόν ο αλγόριθμος έχει το ίδιο επίπεδο κβαντικής ασφάλειας με το «κλασσικό».

Κεφάλαιο 7
Μέτρηση απόδοσης του SATURNIN

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τα αποτελέσματα της σύγκρισης απόδοσης του AES και του Saturnin. Για να μπορέσουμε να έχουμε ένα αντιπροσωπευτικό περιβάλλον για τις μετρήσεις μας, χρησιμοποιούμε ένα IoT board με περιορισμένη υπολογιστική ισχύ, με μικρό μέγεθος μνήμης RAM και με δυνατότητα σύνδεσης στο Διαδίκτυο. Η μικροσυσκευή που χρησιμοποιήσαμε είναι το ESP8266 board και θα αναφερθούμε αναλυτικά σε αυτήν παρακάτω. Η συσκευή αυτή θα έχει το ρόλο του server και θα απαντά σε αιτήσεις του client, στέλνοντάς του κρυπτογραφημένα μηνύματα. Ο client, είναι μια desktop εφαρμογή που τρέχει σε Windows (αν και μπορεί να τρέξει και σε άλλα λειτουργικά συστήματα) και κάνει αιτήσεις για δεδομένα μέσω δικτύου στο ESP8266 και χρονομετρά το συνολικό χρόνο κρυπτογράφησης και αποστολής του κρυπτοκειμένου.

Διαλέξαμε τον AES CTR με κλειδί 256bit, γιατί είναι ο καλύτερος τρόπος λειτουργίας του AES. Θεωρείται, λόγω του μεγέθους του κλειδιού, ασφαλής ακόμα και σε μετακβαντική κρυπτανάλυση. Θα τον συγκρίνουμε σε απόδοση με τον Saturnin Short. Ο Saturnin Short είναι υλοποίηση του Saturnin CTR για μικροσυσκευές όταν τα μηνύματα που θέλουμε να στείλουμε είναι μικρά.

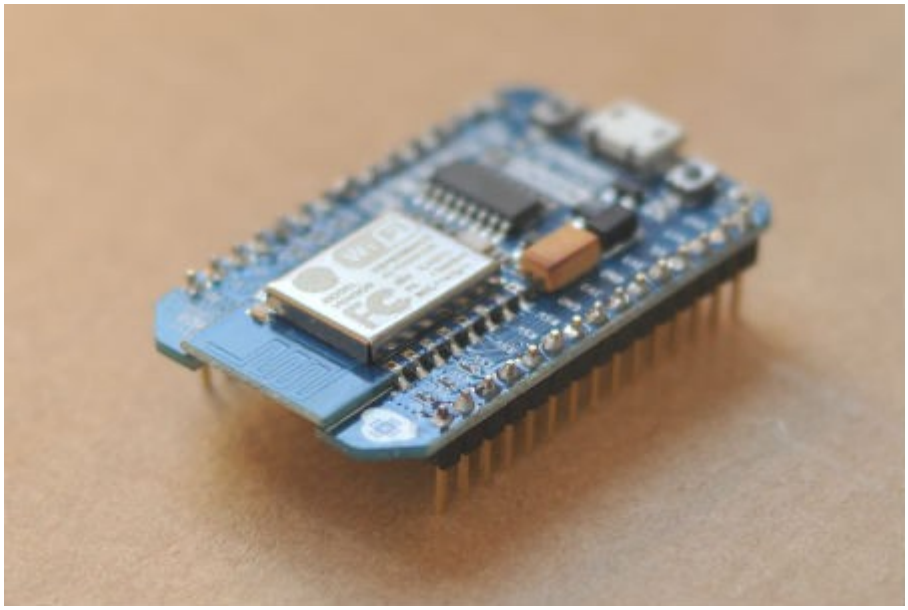
Στη συνέχεια παρουσιάζονται το περιβάλλον ανάπτυξης, οι αντίστοιχες εφαρμογές και τελικά, τα αποτελέσματα των μετρήσεων.

Το περιβάλλον δοκιμής αποτελείται από δύο εφαρμογές. Η μια είναι desktop windows εφαρμογή υλοποιημένη σε περιβάλλον QT και είναι γραμμένη σε C++. Η άλλη είναι υλοποιημένη σε Arduino framework, επίσης σε C++ και τρέχει στο ESP8266.

7.1 Το ESP 8266 Board

Για την μελέτη της απόδοσης του SATURNIN σε lightweight περιβάλλοντα επιλέξαμε το ESP8266 System on a Chip (SoC) και συγκεκριμένα το Node MCU 0.9. Η επίσημη ιστοσελίδα του κατασκευαστή είναι η http://www.nodemcu.com/index_en.html

Το ESP8266, κατασκευάζεται από την Κινέζικη εταιρία Espressif (https://www.espressif.com). Η κεντρική μονάδα επεξεργασίας είναι ο Tensilica L106 32-bit micro controller(MCU). Παρότι είναι εξαιρετικά μικρό σε μέγεθος, περιλαμβάνει και ένα Wi-Fi transceiver. Έχει 11 GPIO pins(General Purpose Input/output pins), και μία αναλογική είσοδο (analog input). Το NodeMCU επιπλέον, διαθέτει και μία Micro-USB θύρα, μέσω της οποίας μπορεί να πάρει τροφοδοσία, αλλά και να επικοινωνήσει με οποιοδήποτε υπολογιστή μέσω serial terminal. Το NodeMCU φαίνεται στην παρακάτω εικόνα.



Εικόνα 30. Το NodeMCU

Έτσι είναι συμβατό με οποιοδήποτε Arduino board και επιπρόσθετα έχει τη δυνατότητα να επικοινωνεί μέσω Wi-Fi. Άρα μπορεί να συνδέεται σε Wi-Fi network, να λειτουργεί σαν web server, να ανταλλάσσει δεδομένα με smartphone και άλλες μικροσυσκευές και γενικά να έχει πλήρη πρόσβαση σε προϊόντα και υπηρεσίες δικτύου. Είναι από τα πιο δημοφιλή IoT board! Και με τιμή λιγότερο από 2$!

Το ESP8266 έχει έναν Tensilica L106 32-bit RISC processor, ο οποίος χρειάζεται υπερβολικά μικρή ποσότητα ενέργειας για τη λειτουργία του και μπορεί να φτάσει σε ταχύτητα ρολογιού έως 160 MHz. Το firmware καθώς και το Wi-Fi stack αφήνει έως και το 80% της επεξεργαστικής ισχύος του να χρησιμοποιηθεί για εφαρμογές. Η συνολική μνήμη RAM που διαθέτει είναι 80 Kbytes. Ο επεξεργαστής μπορεί να έχει τρία διαφορετικά mode λειτουργίας όσον αφορά την κατανάλωση ενέργειας: Το active mode, το sleep mode και το deep sleep mode. Η εναλλαγή των mode γίνεται αυτόματα , ανάλογα με το φόρτο εργασίας και αποσκοπεί στην μεγιστοποίηση της ζωής της μπαταρίας και του χρόνου αυτόνομης λειτουργίας.

Το board μπορεί να επεκταθεί με διάφορα modules τα οποία ενώνονται στα 11 GPIO pins που διαθέτει. Μπορεί να συνεργαστεί με Bluetooth modules, με διάφορους sensors, με κάμερες και άλλα. Τέλος , επειδή είναι Arduino συμβατό, εκτός των modules που παρέχονται από την κατασκευάστρια εταιρία μπορεί να επικοινωνήσει με οποιαδήποτε επέκταση που είναι σχεδιασμένη για Arduino.

Το Node MCU (ESP8266) είναι ιδανικό για την μελέτη της απόδοσης του SATURNIN σε lightweight περιβάλλοντα. Όχι μόνο έχει πλήρεις δυνατότητες δικτύωσης μέσω TCP/IP χρησιμοποιώντας το Wi-Fi που διαθέτει, αλλά επίσης είναι παράδειγμα υπολογιστικής μηχανής χωρίς πολύ μεγάλη επεξεργαστική ισχύ και με πολύ περιορισμένα resources σε μνήμη RAM.

7.1.1 Περιβάλλον προγραμματισμού του ESP8266

Παρότι θα περιμέναμε ότι το βασικό περιβάλλον προγραμματισμού του ESP8266 θα ήταν κάποιος compiler της C (άλλωστε το board έχει ελάχιστα περισσότερη μνήμη από έναν 8bit υπολογιστή της δεκαετίας του 80!), αυτό δεν ισχύει. Το επίσημο περιβάλλον προγραμματισμού του ESP8266 είναι βασισμένο στη γλώσσα LUA.!

Η LUA είναι μια ισχυρή και απλή «scripting language» που χρησιμοποιείται για πολλές εφαρμογές. Ο μεγαλύτερος τομέας χρησιμοποίησής της είναι ο τομέας των παιχνιδιών. Το ESP8266 στο default firmware του φορτώνει έναν «Lua interpreter».

Στην ίδια λογική του εύχρηστου και εύκολου στη μάθηση interpreter το ESP8266 μπορεί να φορτώσει και εναλλακτικά firmware κατασκευασμένα από τρίτους δημιουργούς. Το πιο ενδιαφέρον από όλα είναι το firmware της MicroPython (https://micropython.org/). Με αυτό, το ESP8266 αποκτά ένα πλήρη Python 3 περιβάλλον και μπορεί να εκτελέσει προγράμματα γραμμένα σε Python 3 χωρίς καμία δυσκολία.!

Για την μελέτη της απόδοσης του SATURNIN όμως δεν θα μπορούσαμε να χρησιμοποιήσουμε ένα περιβάλλον ανάπτυξης βασισμένο σε interpreter. Θα πρέπει να προσπαθήσουμε να εκμεταλλευτούμε την επεξεργαστική ισχύ του board στο μεγαλύτερο διαθέσιμο βαθμό και έτσι θα πρέπει να χρησιμοποιήσουμε κάποιο είδος compiler.

Κάθε επεξεργαστής εκτός από τον assembler του, διαθέτει συνήθως και ένα C compiler από την κατασκευάστρια εταιρία. Ο Tensilica L106 είναι αρχικό δημιούργημα της εταιρίας Xtensa, η οποία διαθέτει ένα C compiler που διανέμεται δωρεάν μέσω Github (https://github.com/jcmvbkbc/xtensa-toolchain-build). Έτσι μπορούμε να εκμεταλλευτούμε πλήρως τις δυνατότητες του επεξεργαστή. Δεν

υπάρχουν όμως προεγκατεστημένες απαραίτητες βιβλιοθήκες για επικοινωνία στο Διαδίκτυο, που είναι αναγκαίες για την δοκιμή που θέλουμε.

Για να δημιουργήσει ολοκληρωμένα περιβάλλοντα ανάπτυξης , που μπορούν να χρησιμοποιηθούν από τον αντίστοιχο C compiler, η Espressif διαθέτει στους προγραμματιστές τις παρακάτω επιλογές:

- ESP8266 RTOS SDK: Ολοκληρωμένο περιβάλλον ανάπτυξης βασισμένο στο FreeRTOS (https://freertos.org/). Είναι ένα real time OS για microcontrollers
- ESP8266 Non-OS SDK: Ολοκληρωμένο περιβάλλον ανάπτυξης με όλα τα απαραίτητα api για την ανάπτυξη εφαρμογών. Δεν είναι όμως λειτουργικό σύστημα.
- Simba: Είναι ένα real time OS για microcontrollers το οποίο είναι εξαιρετικά φορητό και απλό στη χρήση.
- Arduino: Είναι το standard framework για microcontrollers. Έχει εκατοντάδες βιβλιοθήκες και υποστηρίζει χιλιάδες διαφορετικά boards. Ο κώδικας των προγραμμάτων τρέχει χωρίς αλλαγές σε συσκευές με τα ίδια χαρακτηριστικά , ανεξαρτήτως microcontroller που διαθέτουν. Ο compiler που διαθέτει είναι συμβατός με C++ κώδικα.

Για την συγγραφή του κώδικα που θα δοκιμάζει τον αλγόριθμο SATURNIN, διαλέξαμε το Arduino Framework. Στην επόμενη παράγραφο θα αναλύσουμε τα χαρακτηριστικά του.

7.1.2 Το «Arduino Framework»

Το Arduino είναι μια «ανοιχτού κώδικα» πλατφόρμα προτυποποίησης ηλεκτρονικών (https://www.arduino.cc/). Η «ιδέα» του Arduino βασίστηκε αρχικά στο μικροελεγκτή ATmega, που είναι ένα μικρό κύκλωμα με επεξεργαστή, μνήμη και προσαρμόσιμες εισόδους/εξόδους. Ουσιαστικά λειτουργεί σαν ένας μικρός, πολύ απλός υπολογιστής, ο οποίος όμως έχει πάρα πολλές δυνατότητες.

Ουσιαστικά το Arduino είναι ένας συνδυασμός υλικού και λογισμικού. Σήμερα υπάρχουν εκατοντάδες διαφορετικές μικροσυσκευές με διαφορετικούς μικροεπεξεργαστές οι οποίες μπορούν να προγραμματιστούν με τα ίδια προγραμματιστικά εργαλεία, χρησιμοποιώντας τις ίδιες βιβλιοθήκες. Από πλευράς λογισμικού, αυτό είναι δυνατό με τη χρήση του Arduino Framework που αποτελείται βασικά από τα παρακάτω:

- C/C++ framework για microcontrollers. Η γλώσσα προγραμματισμού που χρησιμοποιείται είναι το Wiring, που είναι παραλλαγή της C/C++, επομένως η δομή της είναι σε μεγάλο βαθμό

γνωστή. Ο κώδικας που παράγεται χρησιμοποιεί το C compiler της κάθε πλατφόρμας, οπότε η τελική εφαρμογή τρέχει σε μέγιστη δυνατή ταχύτητα.

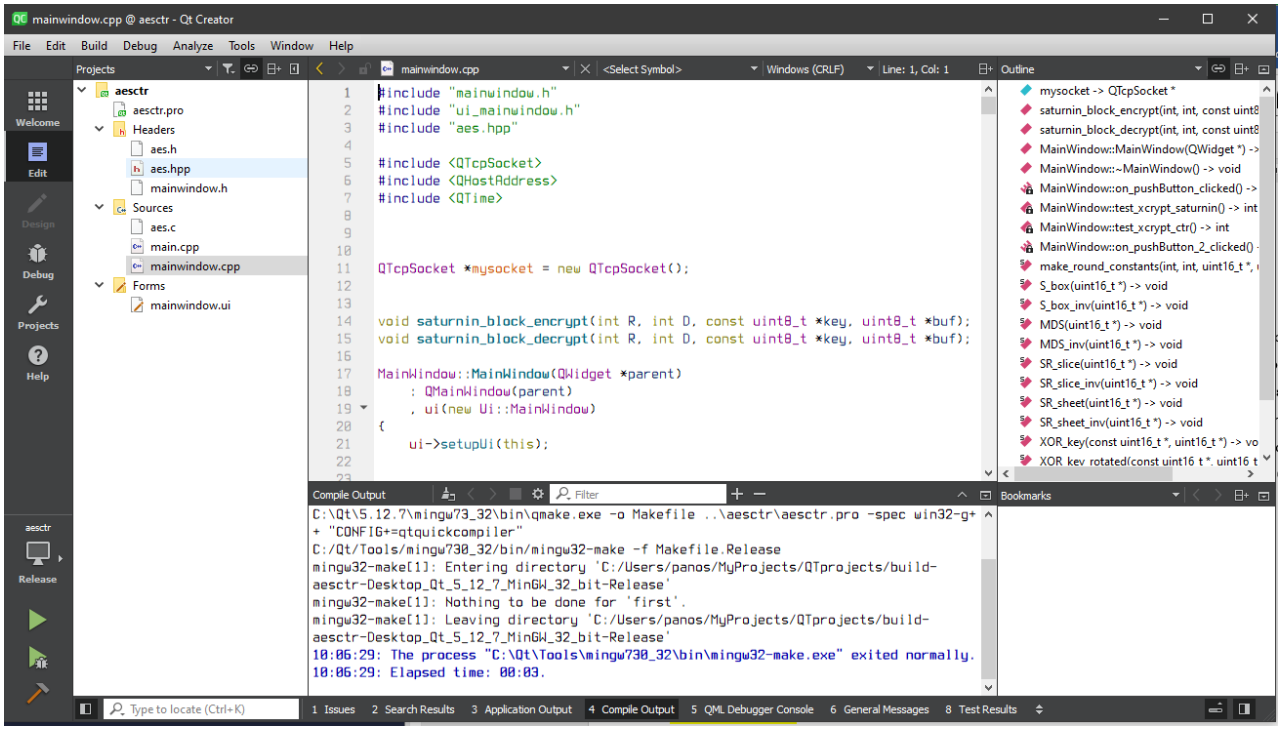
- Device Bootloader: Είναι ένα πρόγραμμα που έρχεται προεγκαταστημένο στο Arduino Framework και «φορτώνει» το πρόγραμμά μας στη μνήμη κατά τη διαδικασία αρχικοποίησης του microcontroller. Αφού ολοκληρωθεί το «φόρτωμα» ξεκινά την εκτέλεση της εφαρμογής.

Οι βιβλιοθήκες που διατίθενται για το Arduino Framework καλύπτουν όλο το φάσμα εφαρμογών και επειδή το Framework είναι το ίδιο για όλους τους microcontroller ανεξαρτήτως επεξεργαστή, μια εφαρμογή που είναι φτιαγμένη για κάποιο συγκεκριμένο board, μπορεί με ευκολία να μεταφερθεί σε ένα διαφορετικό με ελάχιστες αλλαγές στον κώδικα. Π.χ. στην εφαρμογή για το ESP8266 χρησιμοποιήσαμε τη βιβλιοθήκη «crypto» για να κάνουμε κρυπτογράφηση σε AES.

7.2 Περιγραφή των εφαρμογών

7.2.1 Ο «πελάτης» (Desktop)

Ο «cipher network client», όπως ονομάζεται η client εφαρμογή μας είναι γραμμένη σε γλώσσα προγραμματισμού C++ χρησιμοποιώντας το Qt framework (https://www.qt.io/). Το Qt framework είναι ένα cross platform framework το οποίο διατίθεται δωρεάν για open source εφαρμογές. Διευκολύνει τη δημιουργία GUI εφαρμογών που μπορούν να μεταφερθούν σε όλα τα γνωστά λειτουργικά συστήματα, χωρίς καμία αλλαγή στον πηγαίο κώδικα. Χρειάζεται απλά ένα recompilation για το λειτουργικό σύστημα που θέλουμε να τρέξει. Το προτεινόμενο περιβάλλον προγραμματισμού (IDE) για την Qt αλλά και για εφαρμογές C++ είναι ο Qt Creator.



Εικόνα 31. Ο Qt Creator

Για την υλοποίηση του AES χρησιμοποιήσαμε τον «Tiny AES» (https://github.com/kokke/tiny-AES-c) που είναι μια υλοποίηση του AES σε portable C. Υποστηρίζει τους τρόπους λειτουργίας AES ECB, CTR and CBC. Συνοπτικά το api που προσφέρει είναι το παρακάτω:

/* Initialize context calling one of: */
void AES_init_ctx(struct AES_ctx* ctx, const uint8_t* key);
void AES_init_ctx_iv(struct AES_ctx* ctx, const uint8_t* key, const uint8_t* iv);

/* ... or reset IV at random point: */
void AES_ctx_set_iv(struct AES_ctx* ctx, const uint8_t* iv);

/* Then start encrypting and decrypting with the functions below: */
void AES_ECB_encrypt(const struct AES_ctx* ctx, uint8_t* buf);
void AES_ECB_decrypt(const struct AES_ctx* ctx, uint8_t* buf);

void AES_CBC_encrypt_buffer(struct AES_ctx* ctx, uint8_t* buf, uint32_t length);
void AES_CBC_decrypt_buffer(struct AES_ctx* ctx, uint8_t* buf, uint32_t length);

/* Same function for encrypting as for decrypting in CTR mode */
void AES_CTR_xcrypt_buffer(struct AES_ctx* ctx, uint8_t* buf, uint32_t length);

Μετά την αρχικοποίηση, χρησιμοποιούμε τον AES CTR με 256bit κλειδί μέσω της συνάρτησης AES_CTR_xcrypt_buffer. Δεν διαλέξαμε κάποια βιβλιοθήκη που προσφέρει το λειτουργικό σύστημα γιατί θέλαμε να είναι υλοποίηση καθαρά λογισμικού και να μην χρησιμοποιεί δυνατότητες κρυπτογράφησης AES που μπορεί να προσφέρει το υλικό. Έτσι η μέτρηση της διαφοράς στην απόδοση είναι πιο ακριβής.

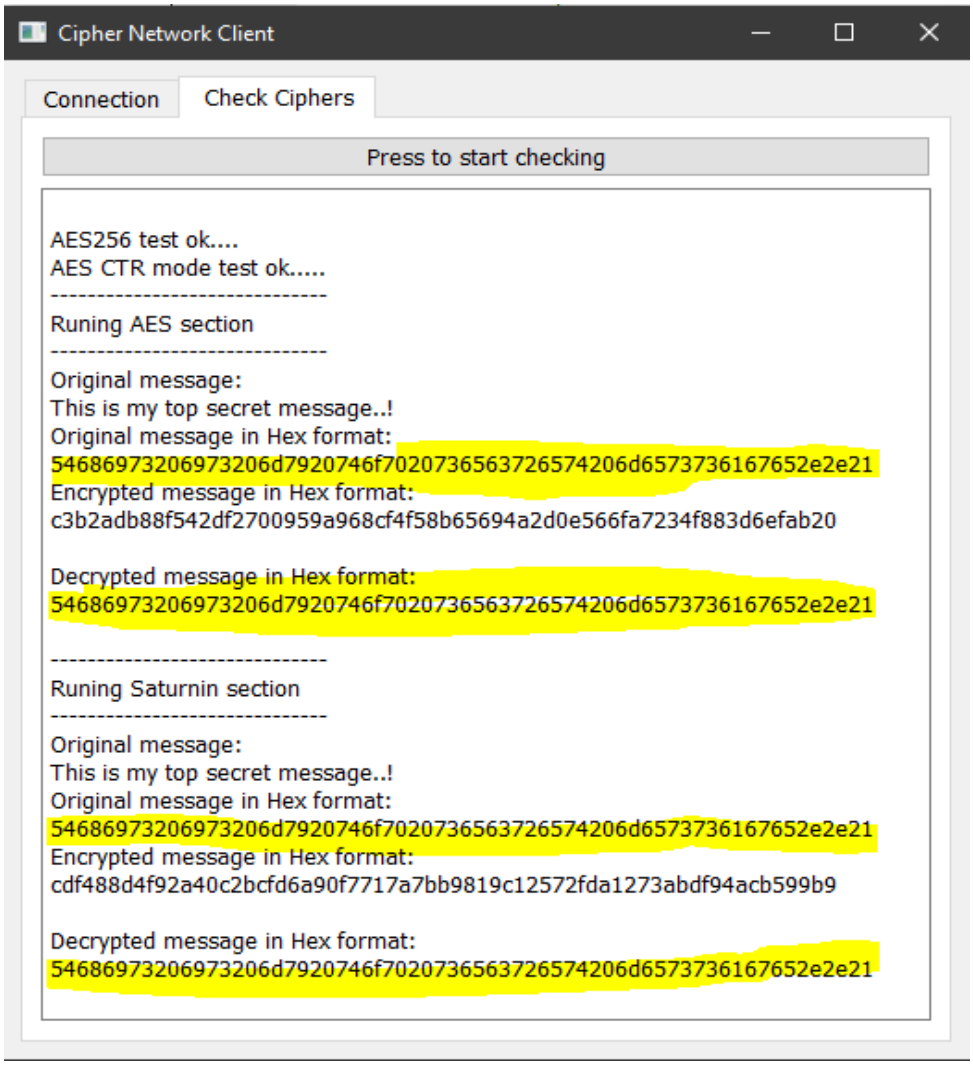
Για τον αλγόριθμο Saturnin, χρησιμοποιήσαμε την υλοποίηση που προτείνουν οι δημιουργοί του και ειδικά την υλοποίηση που είναι γραμμένη σε portable C. Αυτή μας προσφέρει δύο συναρτήσεις για κρυπτογράφηση και αποκρυπτογράφηση και συγκεκριμένα τις:

```
void saturnin_block_encrypt(int R, int D, const uint8_t *key, uint8_t *buf);
void saturnin_block_decrypt(int R, int D, const uint8_t *key, uint8_t *buf);
```

με παραμέτρους τον αριθμό των «υπερ-γύρων»(0-31), τον αριθμό που προσδιορίζει το separation domain (0-15), το κλειδί (32 bytes) και το μπλοκ που θέλουμε να κρυπτογραφήσουμε ή να αποκρυπτογραφήσουμε.

Και εδώ προτιμήσαμε την υλοποίηση σε portable C για να μη χρησιμοποιηθούν δυνατότητες του υλικού που θα μπορούσαν να αυξήσουν την απόδοση.

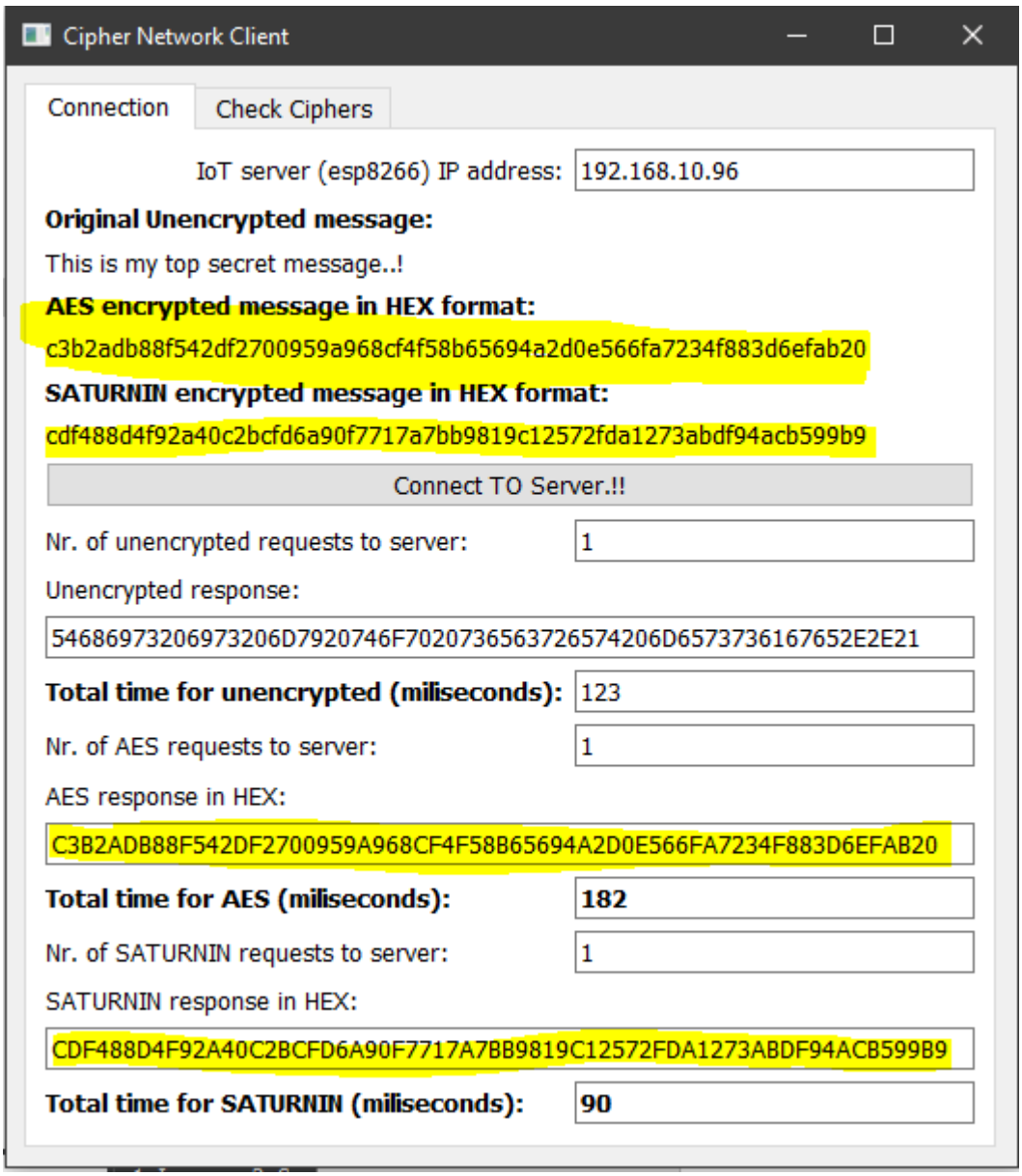
Αρχικά η Desktop εφαρμογή μας δίνει τη δυνατότητα να τσεκάρουμε ότι η υλοποίηση των αλγορίθμων δίνει σωστά αποτελέσματα κατά τη διάρκεια της κρυπτογράφησης και αποκρυπτογράφησης όπως φαίνεται στην παρακάτω εικόνα:



Εικόνα 32. Τα αρχικά μηνύματα και στον AES και στον SATURNIN είναι ίδια με τα αντίστοιχα αποκρυπτογραφημένα

Τέλος, πατώντας το «connect to server», η εφαρμογή επικοινωνεί με το ESP8266 μέσω Wi-Fi και δέχεται ως απάντηση διαδοχικά ένα μήνυμα χωρίς κρυπτογράφηση, έπειτα το ίδιο μήνυμα με κρυπτογράφηση AES και τέλος, πάλι το ίδιο μήνυμα με κρυπτογράφηση Saturnin. Και στις τρεις περιπτώσεις μετρά το συνολικό χρόνο μέχρι την ολοκλήρωση της επικοινωνίας.

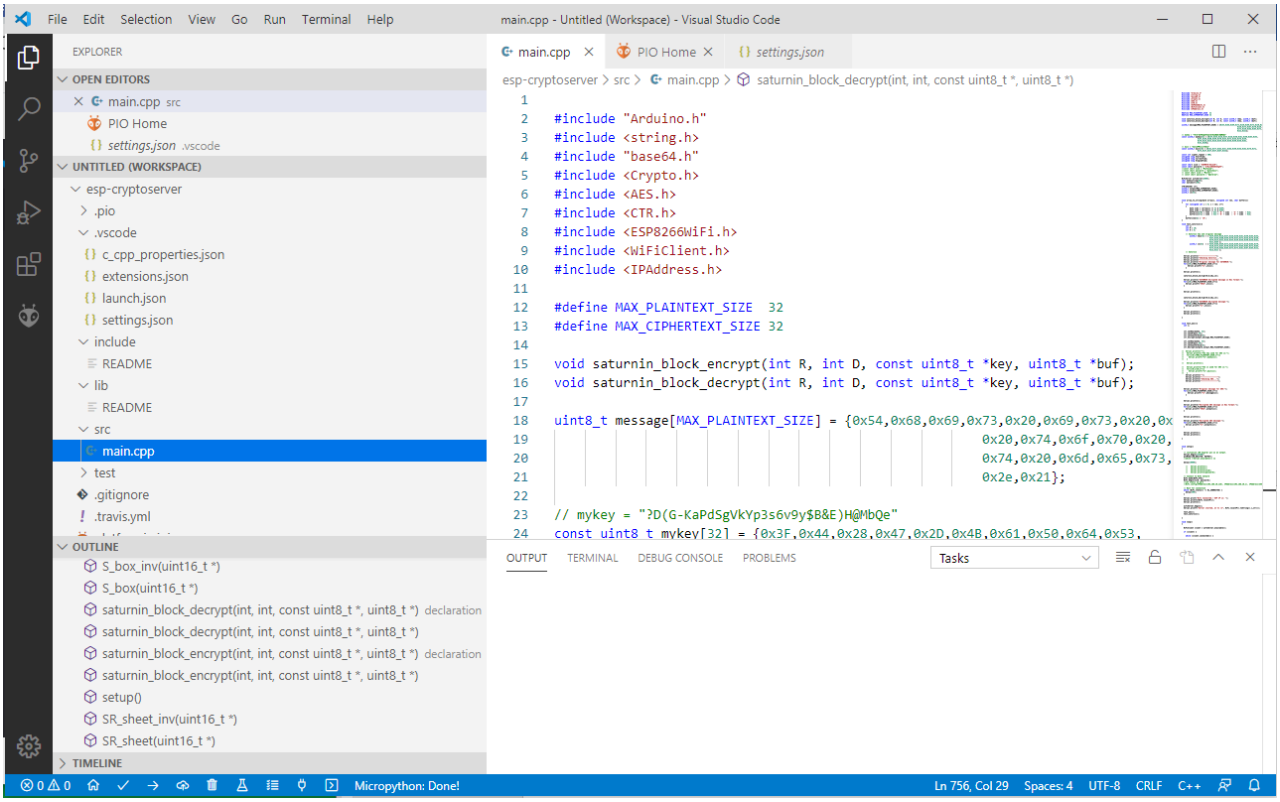
Εφόσον ξέρουμε το αρχικό μήνυμα που είναι το «This is my top secret message..!» και το οποίο έχει μέγεθος 256bit, η εφαρμογή εμφανίζει το αντίστοιχο κρυπτομήνυμα σε AES και Saturnin για να ελέγξουμε ότι το ESP8266 μας έστειλε μέσω δικτύου το σωστό κρυπτοκείμενο. Παρακάτω φαίνεται το «output», όταν στέλνουμε ένα «request» στον server (ESP8266),



Εικόνα 33. Παραλαβή αναμενόμενου κρυπτοκειμένου και καταγραφή χρόνου

7.2.2 Ο «εξυπηρετητής» (ESP8266)

Ο «esp cryptoserver», όπως ονομάζεται η server εφαρμογή μας είναι γραμμένη σε γλώσσα προγραμματισμού C++ χρησιμοποιώντας το Arduino framework Το περιβάλλον προγραμματισμού (IDE) που χρησιμοποιήσαμε είναι το Microsoft Visual Studio Code(https://code.visualstudio.com/). Πρόκειται για ένα open source περιβάλλον ανάπτυξης που υποστηρίζει πολλές γλώσσες προγραμματισμού και τρέχει σε Windows, Linux και Macintosh. Μία εικόνα του περιβάλλοντος φαίνεται παρακάτω:

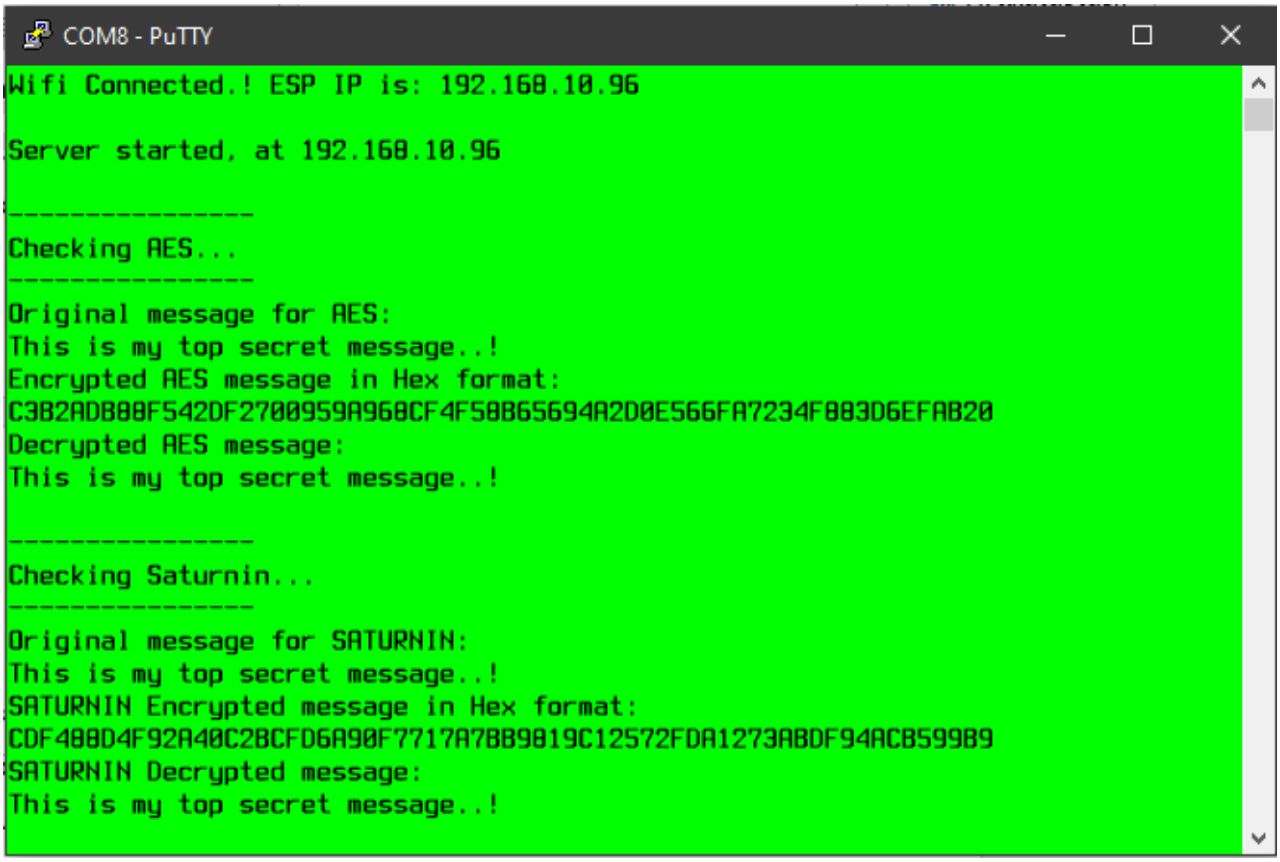


Εικόνα 34. Το Visual Studio Code

Για να μπορέσουμε να προγραμματίσουμε το Arduino framework, χρησιμοποιήσαμε το PlatformIO project (https://platformio.org/). Είναι plugin του Visual Studio Code και επιτρέπει στον χρήστη να διαλέξει μέσα από 35 πλατφόρμες, 20 frameworks και 806 boards (ως τη στιγμή που γράφεται η παρούσα διατριβή). Αφού διαλέξουμε board (το ESP8266 στην περίπτωσή μας), παρουσιάζει όλα τα διαθέσιμα frameworks, διαλέγουμε το επιθυμητό, το οποίο και εγκαθιστά στον υπολογιστή. Μπορούμε έτσι να προγραμματίσουμε, να κάνουμε compile και να φορτώσουμε την εφαρμογή στο board μέσα από το ίδιο IDE.!

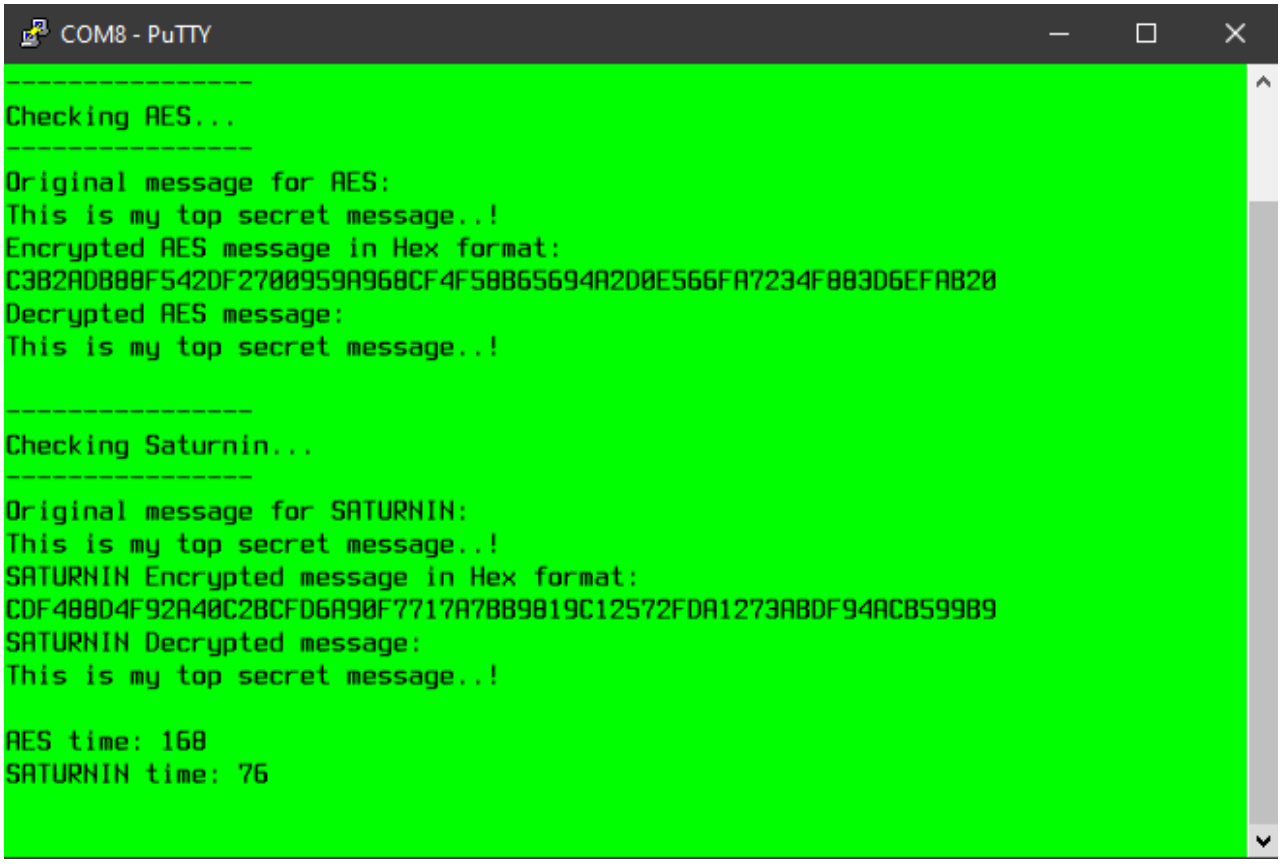
Το Arduino framework περιέχει δύο κύριες συναρτήσεις:

- **void setup()** : Είναι η πρώτη συνάρτηση που «τρέχει» στο board και εκεί κάνουμε όλες τις αρχικοποιήσεις. Εκεί υπάρχει ο κώδικας για τη σύνδεση στο Διαδίκτυο μέσω Wi-Fi, η έναρξη της λειτουργίας του board ως server και κάνουμε και όλες τις αρχικοποιήσεις που χρειάζονται σε μεταβλητές για την ορθή λειτουργία. Τέλος, τσεκάρει την ορθή εκτέλεση των αλγορίθμων και εμφανίζει τα αντίστοιχα μηνύματα στο serial terminal. Η αρχική οθόνη μετά την εκτέλεση της setup() φαίνεται παρακάτω:



Εικόνα 35. Οθόνη κατάστασης μετά την setup()

- **void loop():** Είναι η αμέσως επόμενη συνάρτηση και η ….. τελευταία! Εκτελείται συνέχεια (όπως δηλώνει και το όνομά της) και αν με κάποιο condition στο κώδικα φύγουμε από την loop , τότε το πρόγραμμα τερματίζεται και το board πρέπει να κάνει επανεκκίνηση. Σε αυτή τη συνάρτηση θέτουμε τον server να ακούει συνεχώς στη port 2080 και να στέλνει στον client κρυπτομηνύματα χρησιμοποιώντας αρχικά τον AES και στην στη συνέχεια τον Saturnin. Αφού στείλει τα κρυπτομηνύματα, τυπώνει στο serial terminal το χρόνο που χρειάστηκε το ESP8266 για να κάνει τη κρυπτογράφηση σε milisec (υπενθυμίζουμε ότι ο συνολικός χρόνος μαζί με την επιβάρυνση του δικτύου μετριέται από την client εφαρμογή). Η οθόνη στο serial terminal μετά από ένα request φαίνεται παρακάτω.



Εικόνα 36. Εκτύπωση των χρόνων για κρυπτογράφηση AES και Saturnin σε millisec.

Για την κρυπτογράφηση του AES χρησιμοποιούμε την βιβλιοθήκη «crypto» του Arduino framework που είναι διαθέσιμη μέσω GitHub και υποστηρίζει κρυπτογράφηση AES CTR μέσω λογισμικού. Για την κρυπτογράφηση Saturnin χρησιμοποιούμε την ίδια υλοποίηση με αυτή που έχουμε και στον client, η οποία είναι portable C κώδικας και διατίθεται από τους δημιουργούς του αλγόριθμου.

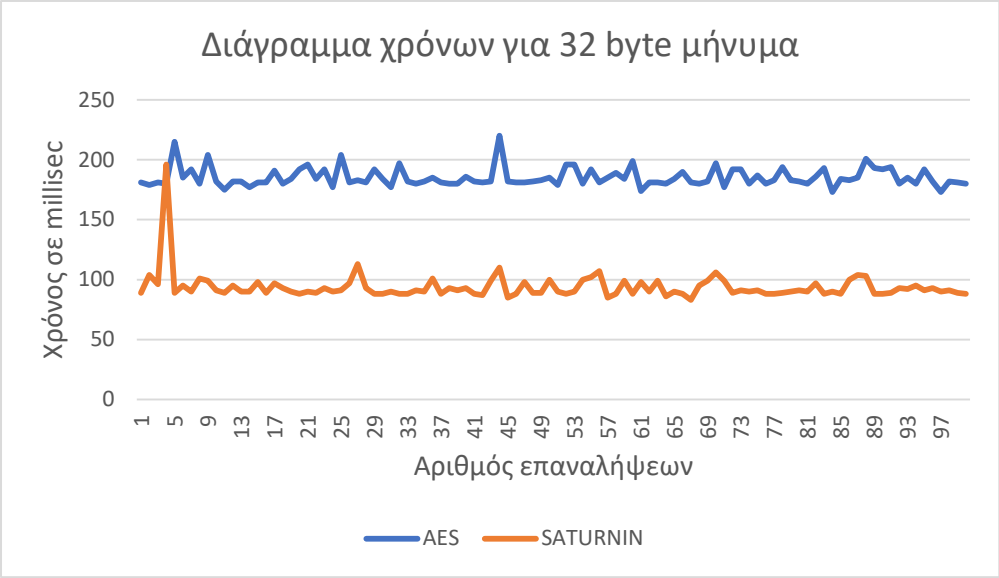
7.3 Αποτελέσματα

Και για τους δύο αλγόριθμους το μέγεθος του κλειδιού είναι 256bit. Το μήνυμα που κρυπτογραφούμε είναι το ANSI string “This is my top secret message..!” το οποίο έχει μέγεθος 32 bytes ή 256 bit. Για να μετρήσουμε την απόδοση , ο client πραγματοποιεί 100 requests από τον server και καταγράφει τους χρόνους που περιλαμβάνουν την αίτηση στον server, την κρυπτογράφηση στον server, την αποστολή του κρυπτοκειμένου πίσω στον client και τελικά την απεικόνιση των δεδομένων στο GUI του client. Ο server στέλνει αρχικά το μήνυμά μας χωρίς κρυπτογράφηση, μετά με κρυπτογράφηση AES και τέλος με κρυπτογράφηση SATURNIN.

Για να έχουμε κάποιες διαχειρίσιμες τιμές, ο server πριν στείλει το κρυπτοκείμενο στον client επαναλαμβάνει τον αλγόριθμο κρυπτογράφησης 500 φορές και στην 501 επανάληψη στέλνει το

κρυπτοκείμενο. Άρα μπορούμε να πούμε ότι κρυπτογραφούμε συνολικά ένα μήνυμα μεγέθους 16032bytes.

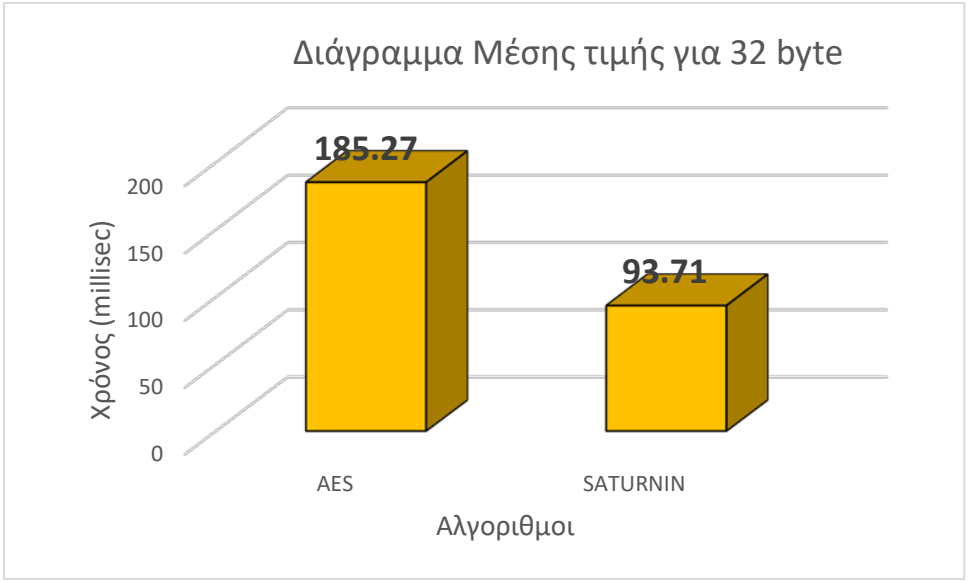
Οι τιμές που καταγράφηκαν στον client φαίνονται στο παρακάτω γράφημα.



Από το γράφημα παρατηρούμε ότι στην συντριπτική πλειοψηφία των περιπτώσεων, ο Saturnin είναι ταχύτερος από τον AES, σχεδόν δύο φορές πιο γρήγορος.

Συγκεκριμένα ο AES κατέγραψε μικρότερο χρόνο ίσο με 173msec και μεγαλύτερο ίσο με 220msec. Στον αντίποδα, ο Saturnin κατέγραψε μικρότερο χρόνο ίσο με 83msec και μεγαλύτερο ίσο με 196msec.

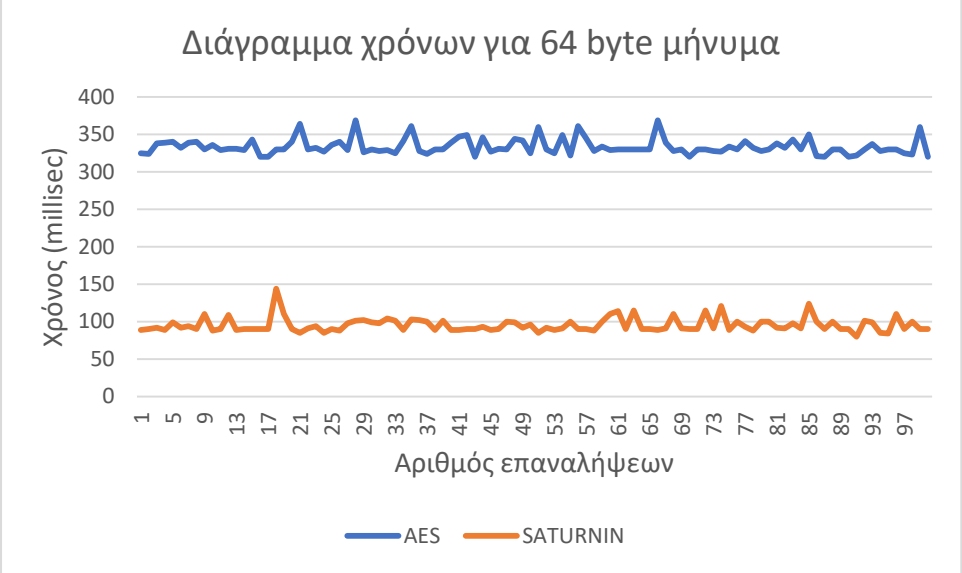
Από το παρακάτω διάγραμμα μέσης τιμής φαίνεται και πάλι ότι ο Saturnin ήταν και κατά μέσο όρο ο πιο γρήγορος αλγόριθμος, με μέση τιμή για τον AES 185,27msec και για τον Saturnin 93,71msec



Παρότι στον client οι χρόνοι διαφέρουν σε κάθε εκτέλεση, στον server, όπου μετράμε μόνο το χρόνο που χρειάζεται το ESP8266 για κρυπτογράφηση, χωρίς τον επιπρόσθετο χρόνο της δικτυακής επικοινωνίας, οι χρόνοι είναι σταθεροί. Συγκεκριμένα σε κάθε request ο AES χρειάζεται 169msec και ο Saturnin 76 msec, με απόκλιση ενός msec σε κάποιες περιπτώσεις. Φαίνεται λοιπόν πως επηρεάζει τη συνολική απόδοση το συνολικό περιβάλλον επικοινωνίας αλλά και πιο συγκεκριμένα οι αλγόριθμοι δικτύου.

Επαναλαμβάνουμε τη δοκιμή, αυξάνοντας το μέγεθος του μηνύματος σε 64 bytes ή 512 bit. Διατηρούμε όλες τις υπόλοιπες παραμέτρους ίδιες, οπότε κατ' αντιστοιχία με τα παραπάνω, κρυπτογραφούμε συνολικά ένα μήνυμα μεγέθους 32064bytes.

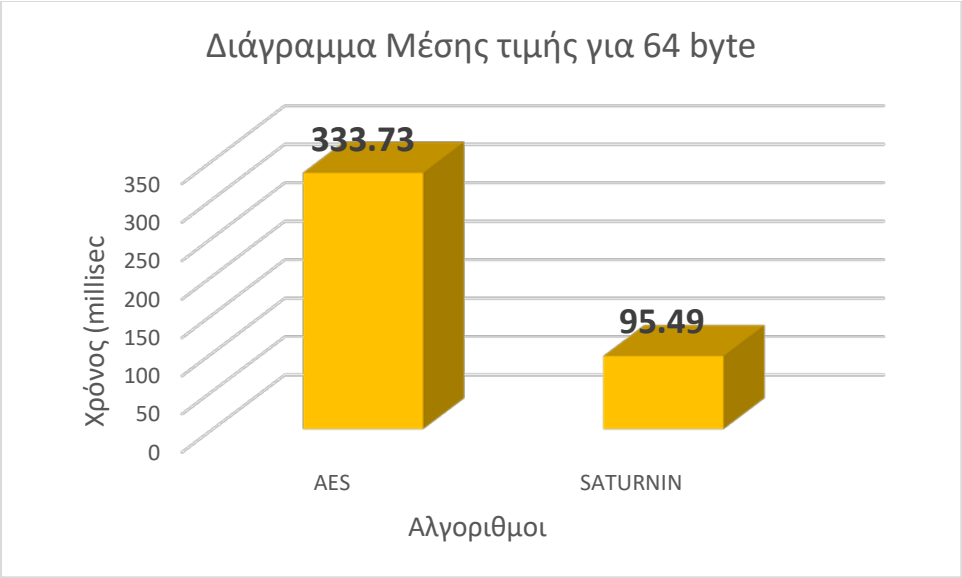
Οι τιμές που καταγράφηκαν στον client φαίνονται στο παρακάτω γράφημα.



Από το γράφημα παρατηρούμε ότι στην συντριπτική πλειοψηφία των περιπτώσεων, ο Saturnin είναι ταχύτερος από τον AES και μάλιστα έως και τρεις φορές πιο γρήγορος.

Συγκεκριμένα ο AES κατέγραψε μικρότερο χρόνο ίσο με 320 msec και μεγαλύτερο ίσο με 369 msec. Στον αντίποδα, ο Saturnin κατέγραψε μικρότερο χρόνο ίσο με 80 msec και μεγαλύτερο ίσο με 144 msec.

Από το παρακάτω διάγραμμα μέσης τιμής φαίνεται και πάλι ότι ο Saturnin ήταν και κατά μέσο όρο ο πιο γρήγορος αλγόριθμος, με μέση τιμή για τον AES 333,73 msec και για τον Saturnin 95,49 msec



Συμπερασματικά, ο Saturnin είναι πιο γρήγορος από τον AES ως block cipher. Τα πολύ καλά αποτελέσματα οφείλονται και στο γεγονός ότι το αρχικό μας μήνυμα ήταν μικρό και μπορούσαμε να χρησιμοποιήσουμε το τρόπο λειτουργίας του Saturnin που αυξάνει την απόδοση όταν το μήνυμα είναι μικρό. Όταν το μήνυμα διπλασιάστηκε, ο Saturnin διατήρησε τις επιδόσεις του ενώ στον AES έχουμε αύξηση σχεδόν 80% στους χρόνους. Βέβαια, όσο μεγαλώνει το μήνυμα, ο Saturnin Short δεν ενδείκνυται για χρήση, γιατί θα παρουσιαστούν ζητήματα ασφάλειας. Σε αυτή την περίπτωση θα πρέπει να χρησιμοποιηθεί ο Saturnin CTR Cascade που είναι περισσότερο απαιτητικός σε υπολογιστική ισχύ από τον Saturnin Short.

Πιστεύουμε ότι και ο Saturnin CTR Cascade, σε λειτουργία block cipher και με μηνύματα μεγάλου μεγέθους θα αποδειχθεί εξίσου γρήγορος και θα υπάρχει σημαντική διαφορά στην απόδοση σε σχέση με τον AES.

Κεφάλαιο 8
Επίλογος

Στην παρούσα διατριβή εστιάσαμε σε μια σημαντική κατηγορία κρυπτογραφικών αλγορίθμων, των λεγόμενων "lightweight". Είδαμε ότι μια νέα γενιά υπολογιστικών συστημάτων, τα «Internet of Things», λόγω της περιορισμένης επεξεργαστικής τους ισχύος χρειάζονται μια νέα γενιά κρυπτογραφικώ αλγορίθμων, οι οποίοι να είναι αποδοτικοί σε περιβάλλοντα περιορισμένης επεξεργαστικής ισχύς. Αναλύσαμε τα «Internet of Things» και αναφέραμε τις απειλές που μπορεί να αντιμετωπίσουν κατά την ανταλλαγή δεδομένων μέσω Διαδικτύου.

Για τους παραπάνω αλγορίθμους, αναφερθήκαμε στα ζητήματα ασφάλειας που μπορεί να προκύψουν σε σχέση με τον τρόπο υλοποίησης τους και τις παραδοχές που έχουν κάνει οι δημιουργοί τους σχετικά. Στα θέματα ασφάλειας συμπεριλάβαμε επίσης και τις μελλοντικές απειλές που θα προκύψουν από την έλευση των κβαντικών υπολογιστών.

Αναφερθήκαμε στο τρέχοντα διαγωνισμό του NIST , που θα καταλήξει στην αποδοχή ενός συνόλου "lightweight" κρυπτογραφικών αλγορίθμων, οι οποίοι θα αποτελέσουν πρότυπα για την ασφαλή επικοινωνία μέσω Διαδικτύου. Αναλύσαμε την οικογένεια αλγορίθμων Saturnin, που είναι υποψήφιοι για τον παραπάνω διαγωνισμό και τους συγκρίναμε με τον αλγόριθμο AES , με τον οποίο μοιράζονται βασικές αρχές και τον βελτιώνουν σε πολλά σημεία.

Κάναμε μια δοκιμή απόδοσης μεταξύ του AES και του Saturnin. Για τον σκοπό αυτό χρησιμοποιήσαμε board με περιορισμένη υπολογιστική ισχύ και αναπτύξαμε το αντίστοιχο λογισμικό. Με την ανταλλαγή μηνυμάτων μεταξύ των εφαρμογών, καταγράψαμε τους χρόνους που χρειάζεται ο κάθε αλγόριθμος, και καταλήξαμε στο συμπέρασμα ότι ο Saturnin είναι αισθητά πιο γρήγορος από τον AES.

Λόγω της απουσίας του Saturnin από τις υπάρχουσες βιβλιοθήκες TLS του Arduino Framework, δεν κατέστη δυνατό να εξεταστεί η συμπεριφορά και η απόδοση του πρωτοκόλλου TLS μεταξύ των εφαρμογών. Μια πρόταση για μελλοντική υλοποίηση θα ήταν η ενσωμάτωση του αλγορίθμου Saturnin στις υπάρχουσες βιβλιοθήκες TLS για το Arduino Framework. Θα ήταν ενδιαφέρον να μπορούσε να

διεξαχθεί μια δοκιμή απόδοσης χρησιμοποιώντας ολόκληρη τη διαδικασία του πρωτοκόλλου TLS και συγκεκριμένα του TLS 1.3. Έτσι θα είχαμε πιο ρεαλιστικά δεδομένα για την απόδοση του αλγορίθμου ως μέρους μιας ολοκληρωμένης σουίτας επικοινωνίας που θα είναι σίγουρα το επόμενα χρόνια το de facto standard ασφαλής επικοινωνίας στο Διαδίκτυο.

Βιβλιογραφία

[1] Claude E. Shannon, «Communication Theory of Secrecy Systems,» *Bell System Technical Journal,* 1949.

[2] Claude E. Shannon, «A Mathematical Theory of Communication,» *Bell System Technical Journal,* 1949.

[3] National Bureau of Standards, «DATA ENCRYPTION STANDARD (DES),» Department of Commerce, 1977.

[4] Whitfield Diffie και Martin E. Hellman, «New Directions in Cryptography,» *IEEE Transactions on Information Theory,* November 1976.

[5] National Institute of Standards and Technology (NIST), «ADVANCED ENCRYPTION STANDARD (AES),» 2001.

[6] A. Menezes, P. van Oorschot και S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[7] Jean-Philippe Aumasson, SERIOUS CRYPTOGRAPHY A Practical Introduction to Modern Encryption, San Francisco: No Starch Press, Inc, 2018.

[8] M. Burmester, Σ. Γκριτζάλης και Σωκράτης Κάτσικας, Σύγχρονη κρυπτογραφία - Θεωρία και εφαρμογές, ΑΘΗΝΑ: Παπασωτηρίου, 2011.

[9] W. Stallings, Cryptography and network security: principles and practice, Boston: Pearson, 2014.

[10] Βασίλειος Κάτος και Γεώργιος Στεφανίδης, Τεχνικές κρυπτογραφιάς και κρυπτανάλυσης, Ζυγός, 2003.

[11] K. Ashton, «That 'Internet of Things' thing,» *RFID Journal,* June 2009.

[12] R. Weinstein, «RFID: a technical overview and its application to the enterprise,» *IT Professional,* 2005.

[13] Oluwatosin Ahmed Amodu και Mohamed Othman , «Machine-to-Machine Communication: An Overview of Opportunities,» *Computer Networks,* November 2018.

[14] Internet Engineering Task Force (IETF) , «The Transport Layer Security (TLS) Protocol Version 1.3,» Internet Engineering Task Force (IETF) , 2018.

[15] F. Hu, Security and Privacy in Internet of Things (IoTs): Models, Algorithms, and Implementations, CRC Press, 2016.

[16] M. Abdur, S. Habib και S. Ullah, «Security Issues in the Internet of Things (IoT): A Comprehensive Study,» *Int. J. Adv. Comput. Sci. Appl.,* 2017.

[17] Lov K. Grover, A fast quantum mechanical algorithm for database search, ACM Press, 1996.

[18] Peter W. Shor, «Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,» AT&T Research, 1995 .

[19] Leslie Xu, «Intel® Advanced Encryption Standard New Instructions (Intel® AES-NI),» Intel Corporation, 2010.

[20] E. Barker και A. Roginsky, «Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths,» National Institute of Standards and Technology, Maryland, 2015.

[21] L. Chen, «Recommendation for Key Derivation through Extraction-then-Expansion,» National Institute of Standards and Technology, Maryland, 2011.

[22] Q. Dang, «Recommendation for Existing Application-Specific Key Derivation Functions,» National Institute of Standards and Technology, Maryland, 2011.

[23] Tunar Sönmez, E. Barker, E. Burr και L. Chen, «Recommendation for Password Based Key Derivation: Part 1: Storage Applications,» National Institute of Standards and Technology, Maryland, 2010.

[24] G. Leander, C. Paar, A. Poschmann και K. Schramm, «New Lightweight DES Variants,» 14th International Workshop on Fast Software Encryption, Luxembourg, 2007.

[25] A. Bogdanov , L. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin και C. Vikkelsoe, «PRESENT: An Ultra-Lightweight Block Cipher,» International Workshop on Cryptographic Hardware and Embedded Systems, Vienna, 2007.

[26] R. Beaulieu, D. Shors, J. Smith και C. Treatman, «SIMON and SPECK Families of Lightweight Block Ciphers,» IACR Cryptology, 2013.

[27] J. Guo, T. Peyrin και A. Poschmann, «The PHOTON Family of Lightweight Hash,» 31st Annual International Cryptology Conference, Santa Barbara, 2011.

[28] J. Aumasson, L. Henzen, W. Meier και M. Naya-Plasencia, «Quark: A Lightweight Hash,» *Journal of Cryptology,* 2013.

[29] A. Bogdanov, M. Knežević και G. Leander, «SPONGENT: A Lightweight Hash Function,» 13th International Workshop on Cryptographic Hardware and Embedded Systems, Nara, 2011.

[30] S. Hirose, K. Ideguchi και H. Kuwakado, «A Lightweight 256-Bit Hash Function for Hardware and Low-End Devices: Lesamnta-LW,» 13th International Conference on Information Security and Cryptology, Seoul, 2010.

[31] N. Mouha, B. Mennink και A. Van Herrewege, «Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers,» International Conference on Selected Areas in Cryptography, Montreal, 2014.

[32] Z. Gong, P. Hartel και S. Nikova, «TuLP: A Family of Lightweight Message Authentication Codes for Body Sensor Networks,» *Journal of Computer Science and Technology,* 2014.

[33] A. Luykx, B. Preneel και E. Tischhauser, «A MAC Mode for Lightweight Block Ciphers,» International Conference on Fast Software Encryption, Bochum, 2016.

[34] ECRYPT, «eSTREAM: the ECRYPT Stream Cipher Project,» 2016.

[35] M. Hell, T. Johansson και W. Meier, «Grain: A Stream Cipher for Constrained Environments,» *International Journal of Wireless and Mobile Computing,* 2007.

[36] Meltem Sönmez Turan, Kerry A. McKay, Çağdaş Çalık, Donghoon Chang και Larry Bassham, «Status Report on the First Round of the NIST Lightweight Cryptography Standardization Process,» National Institute of Standards and Technology, 2019.

[37] C. De Cannière και B. Preneel, «Trivium: 'New Stream Cipher Designs,» Springer, 2008.

[38] S. Babbage και M. Dodd, «The MICKEY Stream Ciphers,» Springer, 2008.

[39] National Institute of Standards and Technology, «Announcing Request for Comments on Lightweight Cryptography Requirements and Evaluation Criteria,» *Federal Register,* May 2018.

[40] M. Sönmez Turan, «NIST Lightweight Cryptography Project - Second round candidates,» National Institute of Standards and Technology, 2019.

[41] A. Canteaut et al, «Saturnin: a suite of lightweight symmetric algorithms for post-quantum security,» 2019.

[42] K. McKay, L. Bassham, M. S. Turan και N. Mouha, «Report on lightweight cryptography,» National Institute of Standards and Technology, Gaithersburg, MD, 2017.

[43] Marc Kaplan, Gaëtan Leuren, Anthony Leverrier και María Naya-Plasencia, «Quantum Differential and Linear Cryptanalysis,» The Information Processing and Communication Laboratory (LTCI), Paris, 2016.

[44] National Institute of Standards and Technology, «SHA-3 Standard:Permutation-Based Hash and Extendable-Output Functions,» U.S. Department of Commerce, 2015.

[45] Xavier Bonnetain, Andre Schrottenloher και Maria Naya-Plasencia, «Quantum security analysis of AES,» IACR Cryptology, 2019.

[46] W. Stallings και Κ. Λιμνιώτης,, Κρυπτογραφία και Ασφάλεια Δικτύων, ΑΘΗΝΑ: Ιων, 2012.

[47] R. Niederhagen και M. Waidner, «Practical Post-Quantum Cryptography,» Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, 2019.

Παράρτημα Α
Πηγαίος κώδικας εφαρμογών

Το περιβάλλον δοκιμής αποτελείται από δύο εφαρμογές. Η μια είναι desktop windows εφαρμογή υλοποιημένη σε περιβάλλον QT και είναι γραμμένη σε C++. Η άλλη είναι υλοποιημένη σε Arduino framework, επίσης σε C++ και τρέχει στο ESP8266

A.1 Windows Desktop Εφαρμογή

- **Αρχείο aesctr.pro**

```
QT       += core gui network

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use
# any Qt feature that has been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt
6.0.0

SOURCES += \
    aes.c \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    aes.h \
    aes.hpp \
    mainwindow.h


FORMS += \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

- **Αρχείο aes.h**

```
#ifndef _AES_H_
#define _AES_H_

#include <stdint.h>

// #define the macros below to 1/0 to enable/disable the mode of operation.
//
// CBC enables AES encryption in CBC-mode of operation.
// CTR enables encryption in counter-mode.
// ECB enables the basic ECB 16-byte block algorithm. All can be enabled simultaneously.

// The #ifndef-guard allows it to be configured before #include'ing or at compile time.
#ifndef CBC
 #define CBC 1
#endif

#ifndef ECB
 #define ECB 1
#endif

#ifndef CTR
 #define CTR 1
#endif


//#define AES128 1
//#define AES192 1
#define AES256 1

#define AES_BLOCKLEN 16 // Block length in bytes - AES is 128b block only

#if defined(AES256) && (AES256 == 1)
   #define AES_KEYLEN 32
   #define AES_keyExpSize 240
#elif defined(AES192) && (AES192 == 1)
   #define AES_KEYLEN 24
   #define AES_keyExpSize 208
#else
   #define AES_KEYLEN 16   // Key length in bytes
   #define AES_keyExpSize 176
#endif

struct AES_ctx
{
 uint8_t RoundKey[AES_keyExpSize];
#if (defined(CBC) && (CBC == 1)) || (defined(CTR) && (CTR == 1))
 uint8_t Iv[AES_BLOCKLEN];
#endif
};

void AES_init_ctx(struct AES_ctx* ctx, const uint8_t* key);
#if (defined(CBC) && (CBC == 1)) || (defined(CTR) && (CTR == 1))
void AES_init_ctx_iv(struct AES_ctx* ctx, const uint8_t* key, const uint8_t* iv);
void AES_ctx_set_iv(struct AES_ctx* ctx, const uint8_t* iv);
#endif

#if defined(ECB) && (ECB == 1)
// buffer size is exactly AES_BLOCKLEN bytes;
// you need only AES_init_ctx as IV is not used in ECB
```

```
// NB: ECB is considered insecure for most uses
void AES_ECB_encrypt(const struct AES_ctx* ctx, uint8_t* buf);
void AES_ECB_decrypt(const struct AES_ctx* ctx, uint8_t* buf);

#endif // #if defined(ECB) && (ECB == !)


#if defined(CBC) && (CBC == 1)
// buffer size MUST be mutile of AES_BLOCKLEN;
// Suggest https://en.wikipedia.org/wiki/Padding_(cryptography)#PKCS7 for padding scheme
// NOTES: you need to set IV in ctx via AES_init_ctx_iv() or AES_ctx_set_iv()
//        no IV should ever be reused with the same key
void AES_CBC_encrypt_buffer(struct AES_ctx* ctx, uint8_t* buf, uint32_t length);
void AES_CBC_decrypt_buffer(struct AES_ctx* ctx, uint8_t* buf, uint32_t length);

#endif // #if defined(CBC) && (CBC == 1)


#if defined(CTR) && (CTR == 1)

// Same function for encrypting as for decrypting.
// IV is incremented for every block, and used after encryption as XOR-compliment for output
// Suggesting https://en.wikipedia.org/wiki/Padding_(cryptography)#PKCS7 for padding scheme
// NOTES: you need to set IV in ctx with AES_init_ctx_iv() or AES_ctx_set_iv()
//        no IV should ever be reused with the same key
void AES_CTR_xcrypt_buffer(struct AES_ctx* ctx, uint8_t* buf, uint32_t length);

#endif // #if defined(CTR) && (CTR == 1)


#endif // _AES_H_
```

- **Αρχείο aes.hpp**

```
#ifndef _AES_HPP_
#define _AES_HPP_

#ifndef __cplusplus
#error Do not include the hpp header in a c project!
#endif //__cplusplus

extern "C" {
#include "aes.h"
}

#endif //_AES_HPP_
```

- **Αρχείο mainwindow.h**

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
```

```
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

private:
    Ui::MainWindow *ui;

    int test_xcrypt_ctr();
    int test_xcrypt_saturnin();
};
#endif // MAINWINDOW_H
```

- **Αρχείο aes.c**

```
/*
This is an implementation of the AES algorithm, specifically ECB, CTR and CBC mode.
Block size can be chosen in aes.h - available choices are AES128, AES192, AES256.
The implementation is verified against the test vectors in:
  National Institute of Standards and Technology Special Publication 800-38A 2001 ED
ECB-AES128
----------
  plain-text:
    6bc1bee22e409f96e93d7e117393172a
    ae2d8a571e03ac9c9eb76fac45af8e51
    30c81c46a35ce411e5fbc1191a0a52ef
    f69f2445df4f9b17ad2b417be66c3710
  key:
    2b7e151628aed2a6abf7158809cf4f3c
  resulting cipher
    3ad77bb40d7a3660a89ecaf32466ef97
    f5d3d58503b9699de785895a96fdbaaf
    43b1cd7f598ece23881b00e3ed030688
    7b0c785e27e8ad3f8223207104725dd4
NOTE:   String length must be evenly divisible by 16byte (str_len % 16 == 0)
        You should pad the end of the string with zeros if this is not the case.
        For AES192/256 the key size is proportionally larger.
*/


/*****************************************************************************/
/* Includes:                                                                 */
/*****************************************************************************/
#include <string.h> // CBC mode, for memset
#include "aes.h"

/*****************************************************************************/
/* Defines:                                                                  */
/*****************************************************************************/
// The number of columns comprising a state in AES. This is a constant in AES. Value=4
#define Nb 4

#if defined(AES256) && (AES256 == 1)
    #define Nk 8
    #define Nr 14
```

```
#elif defined(AES192) && (AES192 == 1)
  #define Nk 6
  #define Nr 12
#else
  #define Nk 4        // The number of 32 bit words in a key.
  #define Nr 10       // The number of rounds in AES Cipher.
#endif

// jcallan@github points out that declaring Multiply as a function
// reduces code size considerably with the Keil ARM compiler.
// See this link for more information: https://github.com/kokke/tiny-AES-C/pull/3
#ifndef MULTIPLY_AS_A_FUNCTION
 #define MULTIPLY_AS_A_FUNCTION 0
#endif



/*****************************************************************************/
/* Private variables:                                                        */
/*****************************************************************************/
// state - array holding the intermediate results during decryption.
typedef uint8_t state_t[4][4];



// The lookup-tables are marked const so they can be placed in read-only storage instead of RAM
// The numbers below can be computed dynamically trading ROM for RAM -
// This can be useful in (embedded) bootloader applications, where ROM is often limited.
static const uint8_t sbox[256] = {
  //0     1    2      3     4    5     6     7      8    9     A      B    C     D     E     F
  0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
  0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
  0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
  0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
  0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
  0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
  0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
  0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
  0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
  0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
  0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
  0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
  0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
  0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
  0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
  0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 };

static const uint8_t rsbox[256] = {
  0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
  0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
  0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
  0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
  0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
  0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
  0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
  0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
  0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
  0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
  0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
  0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
  0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
  0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
  0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
```

```
  0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d };

// The round constant word array, Rcon[i], contains the values given by
// x to the power (i-1) being powers of x (x is denoted as {02}) in the field GF(2^8)
static const uint8_t Rcon[11] = {
  0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36 };

/*
 * Jordan Goulder points out in PR #12 (https://github.com/kokke/tiny-AES-C/pull/12),
 * that you can remove most of the elements in the Rcon array, because they are unused.
 *
 * From Wikipedia's article on the Rijndael key schedule @
https://en.wikipedia.org/wiki/Rijndael_key_schedule#Rcon
 *
 * "Only the first some of these constants are actually used – up to rcon[10] for AES-128 (as 11 round keys are
needed),
 *  up to rcon[8] for AES-192, up to rcon[7] for AES-256. rcon[0] is not used in AES algorithm."
 */


/*****************************************************************************/
/* Private functions:                                                        */
/*****************************************************************************/
/*
static uint8_t getSBoxValue(uint8_t num)
{
  return sbox[num];
}
*/
#define getSBoxValue(num) (sbox[(num)])
/*
static uint8_t getSBoxInvert(uint8_t num)
{
  return rsbox[num];
}
*/
#define getSBoxInvert(num) (rsbox[(num)])

// This function produces Nb(Nr+1) round keys. The round keys are used in each round to decrypt the states.
static void KeyExpansion(uint8_t* RoundKey, const uint8_t* Key)
{
  unsigned i, j, k;
  uint8_t tempa[4]; // Used for the column/row operations

  // The first round key is the key itself.
  for (i = 0; i < Nk; ++i)
  {
    RoundKey[(i * 4) + 0] = Key[(i * 4) + 0];
    RoundKey[(i * 4) + 1] = Key[(i * 4) + 1];
    RoundKey[(i * 4) + 2] = Key[(i * 4) + 2];
    RoundKey[(i * 4) + 3] = Key[(i * 4) + 3];
  }

  // All other round keys are found from the previous round keys.
  for (i = Nk; i < Nb * (Nr + 1); ++i)
  {
    {
      k = (i - 1) * 4;
      tempa[0]=RoundKey[k + 0];
      tempa[1]=RoundKey[k + 1];
      tempa[2]=RoundKey[k + 2];
      tempa[3]=RoundKey[k + 3];

    }
```

```
    if (i % Nk == 0)
    {
      // This function shifts the 4 bytes in a word to the left once.
      // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]

      // Function RotWord()
      {
        const uint8_t u8tmp = tempa[0];
        tempa[0] = tempa[1];
        tempa[1] = tempa[2];
        tempa[2] = tempa[3];
        tempa[3] = u8tmp;
      }

      // SubWord() is a function that takes a four-byte input word and
      // applies the S-box to each of the four bytes to produce an output word.

      // Function Subword()
      {
        tempa[0] = getSBoxValue(tempa[0]);
        tempa[1] = getSBoxValue(tempa[1]);
        tempa[2] = getSBoxValue(tempa[2]);
        tempa[3] = getSBoxValue(tempa[3]);
      }

      tempa[0] = tempa[0] ^ Rcon[i/Nk];
    }
#if defined(AES256) && (AES256 == 1)
    if (i % Nk == 4)
    {
      // Function Subword()
      {
        tempa[0] = getSBoxValue(tempa[0]);
        tempa[1] = getSBoxValue(tempa[1]);
        tempa[2] = getSBoxValue(tempa[2]);
        tempa[3] = getSBoxValue(tempa[3]);
      }
    }
#endif
    j = i * 4; k=(i - Nk) * 4;
    RoundKey[j + 0] = RoundKey[k + 0] ^ tempa[0];
    RoundKey[j + 1] = RoundKey[k + 1] ^ tempa[1];
    RoundKey[j + 2] = RoundKey[k + 2] ^ tempa[2];
    RoundKey[j + 3] = RoundKey[k + 3] ^ tempa[3];
  }
}

void AES_init_ctx(struct AES_ctx* ctx, const uint8_t* key)
{
  KeyExpansion(ctx->RoundKey, key);
}
#if (defined(CBC) && (CBC == 1)) || (defined(CTR) && (CTR == 1))
void AES_init_ctx_iv(struct AES_ctx* ctx, const uint8_t* key, const uint8_t* iv)
{
  KeyExpansion(ctx->RoundKey, key);
  memcpy (ctx->Iv, iv, AES_BLOCKLEN);
}
void AES_ctx_set_iv(struct AES_ctx* ctx, const uint8_t* iv)
{
  memcpy (ctx->Iv, iv, AES_BLOCKLEN);
}
#endif
```

```
// This function adds the round key to state.
// The round key is added to the state by an XOR function.
static void AddRoundKey(uint8_t round, state_t* state, const uint8_t* RoundKey)
{
  uint8_t i,j;
  for (i = 0; i < 4; ++i)
  {
    for (j = 0; j < 4; ++j)
    {
      (*state)[i][j] ^= RoundKey[(round * Nb * 4) + (i * Nb) + j];
    }
  }
}

// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
static void SubBytes(state_t* state)
{
  uint8_t i, j;
  for (i = 0; i < 4; ++i)
  {
    for (j = 0; j < 4; ++j)
    {
      (*state)[j][i] = getSBoxValue((*state)[j][i]);
    }
  }
}

// The ShiftRows() function shifts the rows in the state to the left.
// Each row is shifted with different offset.
// Offset = Row number. So the first row is not shifted.
static void ShiftRows(state_t* state)
{
  uint8_t temp;

  // Rotate first row 1 columns to left
  temp           = (*state)[0][1];
  (*state)[0][1] = (*state)[1][1];
  (*state)[1][1] = (*state)[2][1];
  (*state)[2][1] = (*state)[3][1];
  (*state)[3][1] = temp;

  // Rotate second row 2 columns to left
  temp           = (*state)[0][2];
  (*state)[0][2] = (*state)[2][2];
  (*state)[2][2] = temp;

  temp           = (*state)[1][2];
  (*state)[1][2] = (*state)[3][2];
  (*state)[3][2] = temp;

  // Rotate third row 3 columns to left
  temp           = (*state)[0][3];
  (*state)[0][3] = (*state)[3][3];
  (*state)[3][3] = (*state)[2][3];
  (*state)[2][3] = (*state)[1][3];
  (*state)[1][3] = temp;
}

static uint8_t xtime(uint8_t x)
{
  return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
}
```

```
// MixColumns function mixes the columns of the state matrix
static void MixColumns(state_t* state)
{
  uint8_t i;
  uint8_t Tmp, Tm, t;
  for (i = 0; i < 4; ++i)
  {
    t   = (*state)[i][0];
    Tmp = (*state)[i][0] ^ (*state)[i][1] ^ (*state)[i][2] ^ (*state)[i][3] ;
    Tm  = (*state)[i][0] ^ (*state)[i][1] ; Tm = xtime(Tm);  (*state)[i][0] ^= Tm ^ Tmp ;
    Tm  = (*state)[i][1] ^ (*state)[i][2] ; Tm = xtime(Tm);  (*state)[i][1] ^= Tm ^ Tmp ;
    Tm  = (*state)[i][2] ^ (*state)[i][3] ; Tm = xtime(Tm);  (*state)[i][2] ^= Tm ^ Tmp ;
    Tm  = (*state)[i][3] ^ t ;            Tm = xtime(Tm);  (*state)[i][3] ^= Tm ^ Tmp ;
  }
}

// Multiply is used to multiply numbers in the field GF(2^8)
// Note: The last call to xtime() is unneeded, but often ends up generating a smaller binary
//       The compiler seems to be able to vectorize the operation better this way.
//       See https://github.com/kokke/tiny-AES-c/pull/34
#if MULTIPLY_AS_A_FUNCTION
static uint8_t Multiply(uint8_t x, uint8_t y)
{
  return (((y & 1) * x) ^
       ((y>>1 & 1) * xtime(x)) ^
       ((y>>2 & 1) * xtime(xtime(x))) ^
       ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^
       ((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))); /* this last call to xtime() can be omitted */
  }
#else
#define Multiply(x, y)                                \
      (  ((y & 1) * x) ^                              \
      ((y>>1 & 1) * xtime(x)) ^                       \
      ((y>>2 & 1) * xtime(xtime(x))) ^                \
      ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^         \
      ((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))))   \

#endif

#if (defined(CBC) && CBC == 1) || (defined(ECB) && ECB == 1)
// MixColumns function mixes the columns of the state matrix.
// The method used to multiply may be difficult to understand for the inexperienced.
// Please use the references to gain more information.
static void InvMixColumns(state_t* state)
{
  int i;
  uint8_t a, b, c, d;
  for (i = 0; i < 4; ++i)
  {
    a = (*state)[i][0];
    b = (*state)[i][1];
    c = (*state)[i][2];
    d = (*state)[i][3];

    (*state)[i][0] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c, 0x0d) ^ Multiply(d, 0x09);
    (*state)[i][1] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c, 0x0b) ^ Multiply(d, 0x0d);
    (*state)[i][2] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c, 0x0e) ^ Multiply(d, 0x0b);
    (*state)[i][3] = Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply(c, 0x09) ^ Multiply(d, 0x0e);
  }
}


// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
```

```
static void InvSubBytes(state_t* state)
{
  uint8_t i, j;
  for (i = 0; i < 4; ++i)
  {
    for (j = 0; j < 4; ++j)
    {
      (*state)[j][i] = getSBoxInvert((*state)[j][i]);
    }
  }
}

static void InvShiftRows(state_t* state)
{
  uint8_t temp;

  // Rotate first row 1 columns to right
  temp = (*state)[3][1];
  (*state)[3][1] = (*state)[2][1];
  (*state)[2][1] = (*state)[1][1];
  (*state)[1][1] = (*state)[0][1];
  (*state)[0][1] = temp;

  // Rotate second row 2 columns to right
  temp = (*state)[0][2];
  (*state)[0][2] = (*state)[2][2];
  (*state)[2][2] = temp;

  temp = (*state)[1][2];
  (*state)[1][2] = (*state)[3][2];
  (*state)[3][2] = temp;

  // Rotate third row 3 columns to right
  temp = (*state)[0][3];
  (*state)[0][3] = (*state)[1][3];
  (*state)[1][3] = (*state)[2][3];
  (*state)[2][3] = (*state)[3][3];
  (*state)[3][3] = temp;
}
#endif // #if (defined(CBC) && CBC == 1) || (defined(ECB) && ECB == 1)

// Cipher is the main function that encrypts the PlainText.
static void Cipher(state_t* state, const uint8_t* RoundKey)
{
  uint8_t round = 0;

  // Add the First round key to the state before starting the rounds.
  AddRoundKey(0, state, RoundKey);

  // There will be Nr rounds.
  // The first Nr-1 rounds are identical.
  // These Nr rounds are executed in the loop below.
  // Last one without MixColumns()
  for (round = 1; ; ++round)
  {
    SubBytes(state);
    ShiftRows(state);
    if (round == Nr) {
      break;
    }
    MixColumns(state);
    AddRoundKey(round, state, RoundKey);
  }
  // Add round key to last round
```

```
  AddRoundKey(Nr, state, RoundKey);
}

#if (defined(CBC) && CBC == 1) || (defined(ECB) && ECB == 1)
static void InvCipher(state_t* state, const uint8_t* RoundKey)
{
  uint8_t round = 0;

  // Add the First round key to the state before starting the rounds.
  AddRoundKey(Nr, state, RoundKey);

  // There will be Nr rounds.
  // The first Nr-1 rounds are identical.
  // These Nr rounds are executed in the loop below.
  // Last one without InvMixColumn()
  for (round = (Nr - 1); ; --round)
  {
    InvShiftRows(state);
    InvSubBytes(state);
    AddRoundKey(round, state, RoundKey);
    if (round == 0) {
      break;
    }
    InvMixColumns(state);
  }

}
#endif // #if (defined(CBC) && CBC == 1) || (defined(ECB) && ECB == 1)

/*****************************************************************************/
/* Public functions:                                                         */
/*****************************************************************************/
#if defined(ECB) && (ECB == 1)


void AES_ECB_encrypt(const struct AES_ctx* ctx, uint8_t* buf)
{
  // The next function call encrypts the PlainText with the Key using AES algorithm.
  Cipher((state_t*)buf, ctx->RoundKey);
}

void AES_ECB_decrypt(const struct AES_ctx* ctx, uint8_t* buf)
{
  // The next function call decrypts the PlainText with the Key using AES algorithm.
  InvCipher((state_t*)buf, ctx->RoundKey);
}


#endif // #if defined(ECB) && (ECB == 1)




#if defined(CBC) && (CBC == 1)


static void XorWithIv(uint8_t* buf, const uint8_t* Iv)
{
  uint8_t i;
  for (i = 0; i < AES_BLOCKLEN; ++i) // The block in AES is always 128bit no matter the key size
  {
    buf[i] ^= Iv[i];
  }
```

```
}

void AES_CBC_encrypt_buffer(struct AES_ctx *ctx, uint8_t* buf, uint32_t length)
{
  uintptr_t i;
  uint8_t *Iv = ctx->Iv;
  for (i = 0; i < length; i += AES_BLOCKLEN)
  {
    XorWithIv(buf, Iv);
    Cipher((state_t*)buf, ctx->RoundKey);
    Iv = buf;
    buf += AES_BLOCKLEN;
  }
  /* store Iv in ctx for next call */
  memcpy(ctx->Iv, Iv, AES_BLOCKLEN);
}

void AES_CBC_decrypt_buffer(struct AES_ctx* ctx, uint8_t* buf,  uint32_t length)
{
  uintptr_t i;
  uint8_t storeNextIv[AES_BLOCKLEN];
  for (i = 0; i < length; i += AES_BLOCKLEN)
  {
    memcpy(storeNextIv, buf, AES_BLOCKLEN);
    InvCipher((state_t*)buf, ctx->RoundKey);
    XorWithIv(buf, ctx->Iv);
    memcpy(ctx->Iv, storeNextIv, AES_BLOCKLEN);
    buf += AES_BLOCKLEN;
  }

}

#endif // #if defined(CBC) && (CBC == 1)



#if defined(CTR) && (CTR == 1)

/* Symmetrical operation: same function for encrypting as for decrypting. Note any IV/nonce should never be
reused with the same key */
void AES_CTR_xcrypt_buffer(struct AES_ctx* ctx, uint8_t* buf, uint32_t length)
{
  uint8_t buffer[AES_BLOCKLEN];

  unsigned i;
  int bi;
  for (i = 0, bi = AES_BLOCKLEN; i < length; ++i, ++bi)
  {
    if (bi == AES_BLOCKLEN) /* we need to regen xor compliment in buffer */
    {

      memcpy(buffer, ctx->Iv, AES_BLOCKLEN);
      Cipher((state_t*)buffer,ctx->RoundKey);

      /* Increment Iv and handle overflow */
      for (bi = (AES_BLOCKLEN - 1); bi >= 0; --bi)
      {
    /* inc will overflow */
        if (ctx->Iv[bi] == 255)
    {
          ctx->Iv[bi] = 0;
          continue;
        }
        ctx->Iv[bi] += 1;
```

```
      break;
     }
     bi = 0;
    }

    buf[i] = (buf[i] ^ buffer[bi]);
  }
}

#endif // #if defined(CTR) && (CTR == 1)
```

- **Αρχείο main.cpp**

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

- **Αρχείο mainwindow.cpp**

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "aes.hpp"

#include <QTcpSocket>
#include <QHostAddress>
#include <QTime>



QTcpSocket *mysocket = new QTcpSocket();


void saturnin_block_encrypt(int R, int D, const uint8_t *key, uint8_t *buf);
void saturnin_block_decrypt(int R, int D, const uint8_t *key, uint8_t *buf);

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);



}

MainWindow::~MainWindow()
{
    delete ui;
```

```
}


void MainWindow::on_pushButton_clicked()
{
   #if defined(AES256)
      ui->textEdit->setText("\nAES256 test ok....");

   #endif
   #if defined(CTR)
      ui->textEdit->append("AES CTR mode test ok.....");
   #endif
   this->test_xcrypt_ctr();
   this->test_xcrypt_saturnin();


}


int MainWindow::test_xcrypt_saturnin(){

   ui->textEdit->append("------------------------------");
   ui->textEdit->append("Runing Saturnin section");
   ui->textEdit->append("------------------------------");
   QString mystring="";
   QByteArray myarray(32,'a');
   int R = 10;
   int D = 6;

   // Saturnin key and original message
      uint8_t key[32] = { 0x3F,0x44,0x28,0x47,0x2D,0x4B,0x61,0x50,0x64,0x53,
                  0x67,0x56,0x6B,0x59,0x70,0x33,0x73,0x36,0x76,0x39,
                  0x79,0x24,0x42,0x26,0x45,0x29,0x48,0x40,0x4D,0x62,
                  0x51,0x65 };
      uint8_t in[32]  = { 0x54,0x68,0x69,0x73,0x20,0x69,0x73,0x20,0x6d,0x79,
                  0x20,0x74,0x6f,0x70,0x20,0x73,0x65,0x63,0x72,0x65,
                  0x74,0x20,0x6d,0x65,0x73,0x73,0x61,0x67,0x65,0x2e,
                  0x2e,0x21 };
   // Saturnin


   ui->textEdit->append("Original message:");
   for(int i=0;i<32;i++){
      QChar mychar = in[i];
      myarray[i] = in[i];
      mystring = mystring + mychar;
    }
   ui->textEdit->append(mystring);
   ui->textEdit->append("Original message in Hex format:");
   ui->textEdit->append(QString(myarray.toHex()));

   saturnin_block_encrypt(R,D,key,in);
   for(int i=0;i<32;i++){
      myarray[i] = in[i];
   }
   ui->textEdit->append("Encrypted message in Hex format:");
   ui->textEdit->append(QString(myarray.toHex()));

   saturnin_block_decrypt(R,D,key,in);
   for(int i=0;i<32;i++){
      myarray[i] = in[i];
   }
   ui->textEdit->append("");
```

```
    ui->textEdit->append("Decrypted message in Hex format:");
    ui->textEdit->append(QString(myarray.toHex()));

    return 0;
}


int MainWindow::test_xcrypt_ctr()
{
    QString mystring="";
    QByteArray myarray(32,'a');
// AES256 key and original message
    uint8_t key[32] = { 0x3F,0x44,0x28,0x47,0x2D,0x4B,0x61,0x50,0x64,0x53,
                        0x67,0x56,0x6B,0x59,0x70,0x33,0x73,0x36,0x76,0x39,
                        0x79,0x24,0x42,0x26,0x45,0x29,0x48,0x40,0x4D,0x62,
                        0x51,0x65 };
    uint8_t in[32]  = { 0x54,0x68,0x69,0x73,0x20,0x69,0x73,0x20,0x6d,0x79,
                        0x20,0x74,0x6f,0x70,0x20,0x73,0x65,0x63,0x72,0x65,
                        0x74,0x20,0x6d,0x65,0x73,0x73,0x61,0x67,0x65,0x2e,
                        0x2e,0x21 };
// aes 256
    uint8_t iv[16]  = { 0x5a,0x77,0x6e,0x4c,0x6d,0x50,0x4d,0x6e,0x78,0x7a,
                        0x73,0x37,0x47,0x47,0x6f,0x32 };


    struct AES_ctx ctx;

    AES_init_ctx_iv(&ctx, key, iv);
    ui->textEdit->append("------------------------------");
    ui->textEdit->append("Runing AES section");
    ui->textEdit->append("------------------------------");
    ui->textEdit->append("Original message:");
    for(int i=0;i<32;i++){
       QChar mychar = in[i];
       myarray[i] = in[i];
       mystring = mystring + mychar;
     }
    ui->textEdit->append(mystring);
    ui->textEdit->append("Original message in Hex format:");
    ui->textEdit->append(QString(myarray.toHex()));

    AES_CTR_xcrypt_buffer(&ctx, in, 32);

    mystring="";
    for(int i=0;i<32;i++){
      myarray[i] = in[i];
     }
     ui->textEdit->append("Encrypted message in Hex format:");
     ui->textEdit->append(QString(myarray.toHex()));
     ui->textEdit->append("");

    AES_init_ctx_iv(&ctx, key, iv);
    AES_CTR_xcrypt_buffer(&ctx, in, 32);

    mystring="";
    for(int i=0;i<32;i++){
       myarray[i] = in[i];
     }
     ui->textEdit->append("Decrypted message in Hex format:");
     ui->textEdit->append(QString(myarray.toHex()));
     ui->textEdit->append("");


    return 0;
```

```
}


void MainWindow::on_pushButton_2_clicked()
{
    QTime myTimer;
    QString server_ip = ui->lineEdit_IP_Address->text();

    int repeat_times = 1;
    myTimer.start();
    int unencMilliseconds =0,
        aesMilliseconds = 0,
        satMilliseconds = 0;

    ui->lineEdit_unenc_request->setText("");
    ui->lineEdit_unenc_response->setText("");
    ui->lineEdit_unenc_time->setText("");
    ui->lineEdit_AES_request->setText("");
    ui->lineEdit_AES_response->setText("");
    ui->lineEdit_AES_time->setText("");
    ui->lineEdit_SAT_request->setText("");
    ui->lineEdit_SAT_response->setText("");
    ui->lineEdit_SAT_time->setText("");

    for(int i=0; i<repeat_times; i++){
        myTimer.restart();
        mysocket->connectToHost(QHostAddress(server_ip), 2080);
        mysocket->write(QByteArray("0"));


        ui->lineEdit_unenc_request->setText(QString::number(i+1));

        mysocket->waitForReadyRead(3000);
        ui->lineEdit_unenc_response->setText(mysocket->readAll());

        mysocket->disconnectFromHost();
        unencMilliseconds = unencMilliseconds + myTimer.elapsed() ;



    }

     ui->lineEdit_unenc_time->setText( QString::number(unencMilliseconds));
     qApp->processEvents();


    for(int i=0; i<repeat_times; i++){
        myTimer.restart();
        mysocket->connectToHost(QHostAddress(server_ip), 2080);
        mysocket->write(QByteArray("1"));

        ui->lineEdit_AES_request->setText(QString::number(i+1));

        mysocket->waitForReadyRead(3000);
        ui->lineEdit_AES_response->setText(mysocket->readAll());

        mysocket->disconnectFromHost();
        aesMilliseconds = aesMilliseconds + myTimer.elapsed() ;

    }
    ui->lineEdit_AES_time->setText( QString::number(aesMilliseconds));
    qApp->processEvents();

    for(int i=0; i<repeat_times; i++){
```

```
        myTimer.restart();
        mysocket->connectToHost(QHostAddress(server_ip), 2080);
        mysocket->write(QByteArray("2"));

        ui->lineEdit_SAT_request->setText(QString::number(i+1));

        mysocket->waitForReadyRead(3000);
        ui->lineEdit_SAT_response->setText(mysocket->readAll());

        mysocket->disconnectFromHost();
        satMilliseconds = satMilliseconds + myTimer.elapsed() ;

    }
    ui->lineEdit_SAT_time->setText( QString::number(satMilliseconds));
    qApp->processEvents();


}


/* ================================================================================ */
/*
 * Saturnin block cipher implementation (reference code, not optimized).
 */

#include <string.h>
#include <stdint.h>

/*
 * Compute round constants for R super-rounds and domain D.
 * Assumptions:
 *   0 <= R <= 31
 *   0 <= D <= 15
 */
static void
make_round_constants(int R, int D, uint16_t *RC0, uint16_t *RC1)
{
    uint16_t x0, x1;
    int n;

    x0 = x1 = D + (R << 4) + 0xFE00;

    for (n = 0; n < R; n ++) {
        int i;

        for (i = 0; i < 16; i ++) {
            x0 = (x0 << 1) ^ (0x2D & -(x0 >> 15));
            x1 = (x1 << 1) ^ (0x53 & -(x1 >> 15));
        }
        RC0[n] = x0;
        RC1[n] = x1;
    }
}

/*
 * Apply the S-boxes on the state (sigma_0 and sigma_1).
 */
static void
S_box(uint16_t *state)
{
    int i;

    for (i = 0; i < 16; i += 8) {
        uint16_t a, b, c, d;
```

```
        /* sigma_0 */
        a = state[i + 0];
        b = state[i + 1];
        c = state[i + 2];
        d = state[i + 3];
        a ^= b & c;
        b ^= a | d;
        d ^= b | c;
        c ^= b & d;
        b ^= a | c;
        a ^= b | d;
        state[i + 0] = b;
        state[i + 1] = c;
        state[i + 2] = d;
        state[i + 3] = a;

        /* sigma_1 */
        a = state[i + 4];
        b = state[i + 5];
        c = state[i + 6];
        d = state[i + 7];
        a ^= b & c;
        b ^= a | d;
        d ^= b | c;
        c ^= b & d;
        b ^= a | c;
        a ^= b | d;
        state[i + 4] = d;
        state[i + 5] = b;
        state[i + 6] = a;
        state[i + 7] = c;
    }
}

/*
 * Apply the inverse S-boxes on the state (inv_sigma_0 and inv_sigma_1).
 */
static void
S_box_inv(uint16_t *state)
{
    int i;

    for (i = 0; i < 16; i += 8) {
        uint16_t a, b, c, d;

        /* inv_sigma_0 */
        b = state[i + 0];
        c = state[i + 1];
        d = state[i + 2];
        a = state[i + 3];
        a ^= b | d;
        b ^= a | c;
        c ^= b & d;
        d ^= b | c;
        b ^= a | d;
        a ^= b & c;
        state[i + 0] = a;
        state[i + 1] = b;
        state[i + 2] = c;
        state[i + 3] = d;

        /* inv_sigma_1 */
        d = state[i + 4];
```

```
        b = state[i + 5];
        a = state[i + 6];
        c = state[i + 7];
        a ^= b | d;
        b ^= a | c;
        c ^= b & d;
        d ^= b | c;
        b ^= a | d;
        a ^= b & c;
        state[i + 4] = a;
        state[i + 5] = b;
        state[i + 6] = c;
        state[i + 7] = d;
    }
}

/*
 * Apply the linear transform (MDS) on the state.
 */
static void
MDS(uint16_t *state)
{
    uint16_t x0, x1, x2, x3, x4, x5, x6, x7;
    uint16_t x8, x9, xa, xb, xc, xd, xe, xf;

    x0 = state[0x0];
    x1 = state[0x1];
    x2 = state[0x2];
    x3 = state[0x3];
    x4 = state[0x4];
    x5 = state[0x5];
    x6 = state[0x6];
    x7 = state[0x7];
    x8 = state[0x8];
    x9 = state[0x9];
    xa = state[0xa];
    xb = state[0xb];
    xc = state[0xc];
    xd = state[0xd];
    xe = state[0xe];
    xf = state[0xf];

#define MUL(t0, t1, t2, t3)   do { \
        uint16_t mul_tmp = (t0); \
        (t0) = (t1); \
        (t1) = (t2); \
        (t2) = (t3); \
        (t3) = mul_tmp ^ (t0); \
    } while (0)

    x8 ^= xc; x9 ^= xd; xa ^= xe; xb ^= xf; /* C ^= D */
    x0 ^= x4; x1 ^= x5; x2 ^= x6; x3 ^= x7; /* A ^= B */
    MUL(x4, x5, x6, x7);                    /* B = MUL(B) */
    MUL(xc, xd, xe, xf);                    /* D = MUL(D) */
    x4 ^= x8; x5 ^= x9; x6 ^= xa; x7 ^= xb; /* B ^= C */
    xc ^= x0; xd ^= x1; xe ^= x2; xf ^= x3; /* D ^= A */
    MUL(x0, x1, x2, x3);                    /* A = MUL(A) */
    MUL(x0, x1, x2, x3);                    /* A = MUL(A) */
    MUL(x8, x9, xa, xb);                    /* C = MUL(C) */
    MUL(x8, x9, xa, xb);                    /* C = MUL(C) */
    x8 ^= xc; x9 ^= xd; xa ^= xe; xb ^= xf; /* C ^= D */
    x0 ^= x4; x1 ^= x5; x2 ^= x6; x3 ^= x7; /* A ^= B */
    x4 ^= x8; x5 ^= x9; x6 ^= xa; x7 ^= xb; /* B ^= C */
    xc ^= x0; xd ^= x1; xe ^= x2; xf ^= x3; /* D ^= A */
```

```
#undef MUL

   state[0x0] = x0;
   state[0x1] = x1;
   state[0x2] = x2;
   state[0x3] = x3;
   state[0x4] = x4;
   state[0x5] = x5;
   state[0x6] = x6;
   state[0x7] = x7;
   state[0x8] = x8;
   state[0x9] = x9;
   state[0xa] = xa;
   state[0xb] = xb;
   state[0xc] = xc;
   state[0xd] = xd;
   state[0xe] = xe;
   state[0xf] = xf;
}

/*
 * Apply the inverse of the linear transform (MDS) on the state.
 */
static void
MDS_inv(uint16_t *state)
{
   uint16_t x0, x1, x2, x3, x4, x5, x6, x7;
   uint16_t x8, x9, xa, xb, xc, xd, xe, xf;

   x0 = state[0x0];
   x1 = state[0x1];
   x2 = state[0x2];
   x3 = state[0x3];
   x4 = state[0x4];
   x5 = state[0x5];
   x6 = state[0x6];
   x7 = state[0x7];
   x8 = state[0x8];
   x9 = state[0x9];
   xa = state[0xa];
   xb = state[0xb];
   xc = state[0xc];
   xd = state[0xd];
   xe = state[0xe];
   xf = state[0xf];

#define MULinv(t0, t1, t2, t3)   do { \
      uint16_t mul_tmp = (t3); \
      (t3) = (t2); \
      (t2) = (t1); \
      (t1) = (t0); \
      (t0) = mul_tmp ^ (t1); \
   } while (0)

   x4 ^= x8; x5 ^= x9; x6 ^= xa; x7 ^= xb; /* B ^= C */
   xc ^= x0; xd ^= x1; xe ^= x2; xf ^= x3; /* D ^= A */
   x8 ^= xc; x9 ^= xd; xa ^= xe; xb ^= xf; /* C ^= D */
   x0 ^= x4; x1 ^= x5; x2 ^= x6; x3 ^= x7; /* A ^= B */
   MULinv(x0, x1, x2, x3);              /* A = MULinv(A) */
   MULinv(x0, x1, x2, x3);              /* A = MULinv(A) */
   MULinv(x8, x9, xa, xb);              /* C = MULinv(C) */
   MULinv(x8, x9, xa, xb);              /* C = MULinv(C) */
   x4 ^= x8; x5 ^= x9; x6 ^= xa; x7 ^= xb; /* B ^= C */
```

```
    xc ^= x0; xd ^= x1; xe ^= x2; xf ^= x3; /* D ^= A */
    MULinv(x4, x5, x6, x7);              /* B = MULinv(B) */
    MULinv(xc, xd, xe, xf);              /* D = MULinv(D) */
    x8 ^= xc; x9 ^= xd; xa ^= xe; xb ^= xf; /* C ^= D */
    x0 ^= x4; x1 ^= x5; x2 ^= x6; x3 ^= x7; /* A ^= B */

#undef MULinv

    state[0x0] = x0;
    state[0x1] = x1;
    state[0x2] = x2;
    state[0x3] = x3;
    state[0x4] = x4;
    state[0x5] = x5;
    state[0x6] = x6;
    state[0x7] = x7;
    state[0x8] = x8;
    state[0x9] = x9;
    state[0xa] = xa;
    state[0xb] = xb;
    state[0xc] = xc;
    state[0xd] = xd;
    state[0xe] = xe;
    state[0xf] = xf;
}

/*
 * Apply the SR_slice permutation.
 */
static void
SR_slice(uint16_t *state)
{
    int i;

    for (i = 0; i < 4; i ++) {
        state[ 4 + i] = ((state[ 4 + i] & 0x7777) << 1)
            | ((state[ 4 + i] & 0x8888) >> 3);
        state[ 8 + i] = ((state[ 8 + i] & 0x3333) << 2)
            | ((state[ 8 + i] & 0xcccc) >> 2);
        state[12 + i] = ((state[12 + i] & 0x1111) << 3)
            | ((state[12 + i] & 0xeeee) >> 1);
    }
}

/*
 * Apply the inverse of the SR_slice permutation.
 */
static void
SR_slice_inv(uint16_t *state)
{
    int i;

    for (i = 0; i < 4; i ++) {
        state[ 4 + i] = ((state[ 4 + i] & 0x1111) << 3)
            | ((state[ 4 + i] & 0xeeee) >> 1);
        state[ 8 + i] = ((state[ 8 + i] & 0x3333) << 2)
            | ((state[ 8 + i] & 0xcccc) >> 2);
        state[12 + i] = ((state[12 + i] & 0x7777) << 1)
            | ((state[12 + i] & 0x8888) >> 3);
    }
}

/*
 * Apply the SR_sheet permutation.
```

```
 */
static void
SR_sheet(uint16_t *state)
{
   int i;

   for (i = 0; i < 4; i ++) {
      state[ 4 + i] = ((state[ 4 + i] <<  4) | (state[ 4 + i] >> 12));
      state[ 8 + i] = ((state[ 8 + i] <<  8) | (state[ 8 + i] >>  8));
      state[12 + i] = ((state[12 + i] << 12) | (state[12 + i] >>  4));
   }
}

/*
 * Apply the inverse of the SR_sheet permutation.
 */
static void
SR_sheet_inv(uint16_t *state)
{
   int i;

   for (i = 0; i < 4; i ++) {
      state[ 4 + i] = ((state[ 4 + i] << 12) | (state[ 4 + i] >>  4));
      state[ 8 + i] = ((state[ 8 + i] <<  8) | (state[ 8 + i] >>  8));
      state[12 + i] = ((state[12 + i] <<  4) | (state[12 + i] >> 12));
   }
}

/*
 * XOR the key into the state.
 */
static void
XOR_key(const uint16_t *key, uint16_t *state)
{
   int i;

   for (i = 0; i < 16; i ++) {
      state[i] ^= key[i];
   }
}

/*
 * XOR the rotated key into the state.
 */
static void
XOR_key_rotated(const uint16_t *key, uint16_t *state)
{
   int i;

   for (i = 0; i < 16; i ++) {
      state[i] ^= (key[i] << 11) | (key[i] >> 5);
   }
}

/*
 * Perform one Saturnin block encryption.
 *   R     number of super-rounds (0 to 31)
 *   D     separation domain (0 to 15)
 *   key   key (32 bytes)
 *   buf   block to encrypt
 * The 'key' and 'buf' buffers may overlap. The encrypted block is
 * written back in 'buf'.
 */
void
```

```
saturnin_block_encrypt(int R, int D, const uint8_t *key, uint8_t *buf)
{
   uint16_t RC0[31], RC1[31];
   uint16_t xk[16], xb[16];
   int i;

   /*
    * Decode key and input block.
    */
   for (i = 0; i < 16; i ++) {
      xk[i] = key[i << 1] + ((uint16_t)key[(i << 1) + 1] << 8);
      xb[i] = buf[i << 1] + ((uint16_t)buf[(i << 1) + 1] << 8);
   }

   /*
    * Compute round constants.
    */
   make_round_constants(R, D, RC0, RC1);

   /*
    * XOR key into state.
    */
   XOR_key(xk, xb);

   /*
    * Run all rounds (two rounds per super-round).
    */
   for (i = 0; i < R; i ++) {
      /*
       * Even round.
       */
      S_box(xb);
      MDS(xb);

      /*
       * Odd round.
       */
      S_box(xb);
      if ((i & 1) == 0) {
         /*
          * Round r = 1 mod 4.
          */
         SR_slice(xb);
         MDS(xb);
         SR_slice_inv(xb);
         xb[0] ^= RC0[i];
         xb[8] ^= RC1[i];
         XOR_key_rotated(xk, xb);
      } else {
         /*
          * Round r = 3 mod 4.
          */
         SR_sheet(xb);
         MDS(xb);
         SR_sheet_inv(xb);
         xb[0] ^= RC0[i];
         xb[8] ^= RC1[i];
         XOR_key(xk, xb);
      }
   }

   /*
    * Encode output block.
    */
```

```
    for (i = 0; i < 16; i ++) {
        buf[(i << 1) + 0] = (uint8_t)xb[i];
        buf[(i << 1) + 1] = (uint8_t)(xb[i] >> 8);
    }
}

/*
 * Perform one Saturnin block decryption.
 *   R     number of super-rounds (0 to 31)
 *   D     separation domain (0 to 15)
 *   key   key (32 bytes)
 *   buf   block to decrypt
 * The 'key' and 'buf' buffers may overlap. The decrypted block is
 * written back in 'buf'.
 */
void
saturnin_block_decrypt(int R, int D, const uint8_t *key, uint8_t *buf)
{
    uint16_t RC0[31], RC1[31];
    uint16_t xk[16], xb[16];
    int i;

    /*
     * Decode key and input block.
     */
    for (i = 0; i < 16; i ++) {
        xk[i] = key[i << 1] + ((uint16_t)key[(i << 1) + 1] << 8);
        xb[i] = buf[i << 1] + ((uint16_t)buf[(i << 1) + 1] << 8);
    }

    /*
     * Compute round constants.
     */
    make_round_constants(R, D, RC0, RC1);

    /*
     * Run all rounds (two rounds per super-round).
     */
    for (i = R - 1; i >= 0; i --) {
        /*
         * Odd round.
         */
        if ((i & 1) == 0) {
            /*
             * Round r = 1 mod 4.
             */
            XOR_key_rotated(xk, xb);
            xb[0] ^= RC0[i];
            xb[8] ^= RC1[i];
            SR_slice(xb);
            MDS_inv(xb);
            SR_slice_inv(xb);
        } else {
            /*
             * Round r = 3 mod 4.
             */
            XOR_key(xk, xb);
            xb[0] ^= RC0[i];
            xb[8] ^= RC1[i];
            SR_sheet(xb);
            MDS_inv(xb);
            SR_sheet_inv(xb);
        }
        S_box_inv(xb);
```

```
        /*
         * Even round.
         */
        MDS_inv(xb);
        S_box_inv(xb);
    }

    /*
     * XOR key into state.
     */
    XOR_key(xk, xb);

    /*
     * Encode output block.
     */
    for (i = 0; i < 16; i ++) {
        buf[(i << 1) + 0] = (uint8_t)xb[i];
        buf[(i << 1) + 1] = (uint8_t)(xb[i] >> 8);
    }
}
```

- **Αρχείο mainwindow.ui**

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>523</width>
    <height>570</height>
   </rect>
  </property>
  <property name="font">
   <font>
    <pointsize>10</pointsize>
   </font>
  </property>
  <property name="windowTitle">
   <string>Cipher Network Client</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="0">
     <layout class="QGridLayout" name="gridLayout_2">
      <item row="0" column="0" rowspan="2" colspan="2">
       <widget class="QTabWidget" name="tabWidget">
        <property name="currentIndex">
         <number>1</number>
        </property>
        <widget class="QWidget" name="tab">
         <attribute name="title">
          <string>Connection</string>
         </attribute>
         <layout class="QGridLayout" name="gridLayout_4">
          <item row="4" column="0" colspan="2">
           <widget class="QLabel" name="label_4">
            <property name="font">
             <font>
```

```
   <pointsize>10</pointsize>
   <weight>75</weight>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
  <string>AES encrypted message in HEX format:</string>
 </property>
</widget>
</item>
<item row="5" column="0" colspan="2">
<widget class="QLabel" name="label_2">
 <property name="text">
  <string>c3b2adb88f542df2700959a968cf4f58b65694a2d0e566fa7234f883d6efab20</string>
 </property>
</widget>
</item>
<item row="0" column="1">
<widget class="QLineEdit" name="lineEdit_IP_Address"/>
</item>
<item row="0" column="0">
<widget class="QLabel" name="label_11">
 <property name="text">
  <string>IoT server (esp8266) IP address:</string>
 </property>
 <property name="alignment">
  <set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>
 </property>
</widget>
</item>
<item row="20" column="0" colspan="2">
<widget class="QLineEdit" name="lineEdit_SAT_response"/>
</item>
<item row="17" column="0">
<widget class="QLabel" name="label_10">
 <property name="font">
  <font>
   <pointsize>10</pointsize>
   <weight>75</weight>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
  <string>Total time for AES (miliseconds):</string>
 </property>
</widget>
</item>
<item row="3" column="0" colspan="2">
<widget class="QLabel" name="label">
 <property name="text">
  <string>This is my top secret message..!</string>
 </property>
</widget>
</item>
<item row="7" column="0" colspan="2">
<widget class="QLabel" name="label_12">
 <property name="font">
  <font>
   <weight>75</weight>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
  <string>SATURNIN encrypted message in HEX format:</string>
```

```
 </property>
 </widget>
</item>
<item row="21" column="1">
 <widget class="QLineEdit" name="lineEdit_SAT_time">
  <property name="font">
   <font>
    <weight>75</weight>
    <bold>true</bold>
   </font>
  </property>
 </widget>
</item>
<item row="11" column="0">
 <widget class="QLabel" name="label_6">
  <property name="text">
   <string>Unencrypted response:</string>
  </property>
 </widget>
</item>
<item row="21" column="0">
 <widget class="QLabel" name="label_16">
  <property name="font">
   <font>
    <weight>75</weight>
    <bold>true</bold>
   </font>
  </property>
  <property name="text">
   <string>Total time for SATURNIN (miliseconds):</string>
  </property>
 </widget>
</item>
<item row="10" column="0">
 <widget class="QLabel" name="label_5">
  <property name="text">
   <string>Nr. of unencrypted requests to server:</string>
  </property>
 </widget>
</item>
<item row="8" column="0" colspan="2">
 <widget class="QLabel" name="label_13">
  <property name="text">
   <string>cdf488d4f92a40c2bcfd6a90f7717a7bb9819c12572fda1273abdf94acb599b9</string>
  </property>
 </widget>
</item>
<item row="19" column="0" colspan="2">
 <widget class="QLabel" name="label_15">
  <property name="text">
   <string>SATURNIN response in HEX:</string>
  </property>
 </widget>
</item>
<item row="14" column="1">
 <widget class="QLineEdit" name="lineEdit_AES_request"/>
</item>
<item row="16" column="0" colspan="2">
 <widget class="QLineEdit" name="lineEdit_AES_response"/>
</item>
<item row="18" column="1">
 <widget class="QLineEdit" name="lineEdit_SAT_request"/>
</item>
<item row="18" column="0">
```

```
 <widget class="QLabel" name="label_14">
  <property name="text">
   <string>Nr. of SATURNIN requests to server:</string>
  </property>
 </widget>
</item>
<item row="12" column="0" colspan="2">
 <widget class="QLineEdit" name="lineEdit_unenc_response"/>
</item>
<item row="17" column="1">
 <widget class="QLineEdit" name="lineEdit_AES_time">
  <property name="font">
   <font>
    <weight>75</weight>
    <bold>true</bold>
   </font>
  </property>
 </widget>
</item>
<item row="2" column="0" colspan="2">
 <widget class="QLabel" name="label_3">
  <property name="font">
   <font>
    <pointsize>10</pointsize>
    <weight>75</weight>
    <bold>true</bold>
   </font>
  </property>
  <property name="text">
   <string>Original Unencrypted message:</string>
  </property>
 </widget>
</item>
<item row="14" column="0">
 <widget class="QLabel" name="label_8">
  <property name="text">
   <string>Nr. of AES requests to server:</string>
  </property>
 </widget>
</item>
<item row="10" column="1">
 <widget class="QLineEdit" name="lineEdit_unenc_request"/>
</item>
<item row="13" column="1">
 <widget class="QLineEdit" name="lineEdit_unenc_time"/>
</item>
<item row="15" column="0">
 <widget class="QLabel" name="label_9">
  <property name="text">
   <string>AES response in HEX:</string>
  </property>
 </widget>
</item>
<item row="13" column="0">
 <widget class="QLabel" name="label_7">
  <property name="font">
   <font>
    <pointsize>10</pointsize>
    <weight>75</weight>
    <bold>true</bold>
   </font>
  </property>
  <property name="text">
   <string>Total time for unencrypted (miliseconds):</string>
```

```
        </property>
       </widget>
      </item>
      <item row="9" column="0" colspan="2">
       <widget class="QPushButton" name="pushButton_2">
        <property name="text">
         <string>Connect TO Server.!!</string>
        </property>
       </widget>
      </item>
     </layout>
    </widget>
    <widget class="QWidget" name="tab_2">
     <attribute name="title">
      <string>Check Ciphers</string>
     </attribute>
     <layout class="QGridLayout" name="gridLayout_3">
      <item row="0" column="0">
       <widget class="QPushButton" name="pushButton">
        <property name="text">
         <string>Press to start checking</string>
        </property>
       </widget>
      </item>
      <item row="1" column="0">
       <widget class="QTextEdit" name="textEdit"/>
      </item>
     </layout>
    </widget>
   </widget>
  </item>
 </layout>
</widget>
</widget>
<resources/>
<connections/>
</ui>
```

A.2 ESP8266 Arduino Εφαρμογή

Τα τμήματα και υπο-τμήματα των παραρτημάτων ακολουθούν αρίθμηση αντίστοιχη με αυτά των

- **Αρχείο main.cpp**

```
#include "Arduino.h"
#include <string.h>
#include "base64.h"
#include <Crypto.h>
#include <AES.h>
#include <CTR.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <IPAddress.h>

#define MAX_PLAINTEXT_SIZE  32
```

```
#define MAX_CIPHERTEXT_SIZE 32

void saturnin_block_encrypt(int R, int D, const uint8_t *key, uint8_t *buf);
void saturnin_block_decrypt(int R, int D, const uint8_t *key, uint8_t *buf);

uint8_t message[MAX_PLAINTEXT_SIZE] = {0x54,0x68,0x69,0x73,0x20,0x69,0x73,0x20,0x6d,0x79,
                                       0x20,0x74,0x6f,0x70,0x20,0x73,0x65,0x63,0x72,0x65,
                                       0x74,0x20,0x6d,0x65,0x73,0x73,0x61,0x67,0x65,0x2e,
                                       0x2e,0x21};

// mykey = "?D(G-KaPdSgVkYp3s6v9y$B&E)H@MbQe"
const uint8_t mykey[32] = {0x3F,0x44,0x28,0x47,0x2D,0x4B,0x61,0x50,0x64,0x53,
          0x67,0x56,0x6B,0x59,0x70,0x33,0x73,0x36,0x76,0x39,
          0x79,0x24,0x42,0x26,0x45,0x29,0x48,0x40,0x4D,0x62,
          0x51,0x65};

// myiv = "ZwnLmPMnxzs7GGo2"
const uint8_t myiv[16] = {0x5a,0x77,0x6e,0x4c,0x6d,0x50,0x4d,0x6e,0x78,0x7a,
          0x73,0x37,0x47,0x47,0x6f,0x32};

const int cipher_repeat = 500;
unsigned long StartTime;
unsigned long CurrentTime;
unsigned long ElapsedTime;

const char* ssid = "COSMOTE-82112E";
const char* password = "jT6jveRXGGhkAgVV";
//const char* ssid = "Moufasam";
//const char* password = "panos3wsx";
// const char* ssid = "Master";
// const char* password = "m@sterae";

WiFiServer wifiServer(2080);
char myhexstring[65];
char mytimestr[40];

CTR<AES256> ctr;
uint8_t output[MAX_CIPHERTEXT_SIZE];
uint8_t output2[MAX_CIPHERTEXT_SIZE];
uint8_t in[32];



void array_to_string(byte array[], unsigned int len, char buffer[])
{
   for (unsigned int i = 0; i < len; i++)
   {
      byte nib1 = (array[i] >> 4) & 0x0F;
      byte nib2 = (array[i] >> 0) & 0x0F;
      buffer[i*2+0] = nib1  < 0xA ? '0' + nib1  : 'A' + nib1  - 0xA;
      buffer[i*2+1] = nib2  < 0xA ? '0' + nib2  : 'A' + nib2  - 0xA;
   }
   buffer[len*2] = '\0';
}

void test_saturnin(){
   int i;
   int R = 10;
   int D = 6;

   // Saturnin key and original message
      uint8_t key[32] = { 0x3F,0x44,0x28,0x47,0x2D,0x4B,0x61,0x50,0x64,0x53,
                  0x67,0x56,0x6B,0x59,0x70,0x33,0x73,0x36,0x76,0x39,
                  0x79,0x24,0x42,0x26,0x45,0x29,0x48,0x40,0x4D,0x62,
```

```
                0x51,0x65 };
    uint8_t in[32]  = { 0x54,0x68,0x69,0x73,0x20,0x69,0x73,0x20,0x6d,0x79,
                0x20,0x74,0x6f,0x70,0x20,0x73,0x65,0x63,0x72,0x65,
                0x74,0x20,0x6d,0x65,0x73,0x73,0x61,0x67,0x65,0x2e,
                0x2e,0x21 };
  // Saturnin

  Serial.println("----------------");
  Serial.println("Checking Saturnin...");
  Serial.println("----------------");
  Serial.println("Original message for SATURNIN:");
  for(i=0;i<MAX_PLAINTEXT_SIZE;i++){
     Serial.printf("%c",in[i]);
   }

  Serial.println();

  saturnin_block_encrypt(R,D,key,in);

  Serial.println("SATURNIN Encrypted message in Hex format:");
  for(i=0;i<MAX_PLAINTEXT_SIZE;i++){
   Serial.printf("%02X",in[i]);
  }


  Serial.println();


  saturnin_block_decrypt(R,D,key,in);

  Serial.println("SATURNIN Decrypted message:");
  for(i=0;i<MAX_PLAINTEXT_SIZE;i++){
   Serial.printf("%c",in[i]);
  }

  Serial.println();
  Serial.println();

}


void test_aes(){
  int i;


  ctr.setKey(mykey, 32);
  ctr.setIV(myiv,16);
  ctr.setCounterSize(4);
  ctr.encrypt(output,message,MAX_PLAINTEXT_SIZE);

  ctr.setKey(mykey, 32);
  ctr.setIV(myiv,16);
  ctr.setCounterSize(4);
  ctr.decrypt(output2,output,MAX_PLAINTEXT_SIZE);

//  Serial.println("");
//   Serial.println("The key used for AES is:");
//   for(i=0;i<MAX_PLAINTEXT_SIZE;i++){
//     Serial.printf("%c",mykey[i]);
//   }

//   Serial.println();

//   Serial.println("The iv used for AES is:");
```

```
//   for(i=0;i<16;i++){
//     Serial.printf("%c",myiv[i]);
//   }
    Serial.println("");
    Serial.println("----------------");
    Serial.println("Checking AES...");
    Serial.println("----------------");



  Serial.println("Original message for AES:");
  for(i=0;i<MAX_PLAINTEXT_SIZE;i++){
     Serial.printf("%c",message[i]);
    }


  Serial.println();

  Serial.println("Encrypted AES message in Hex format:");
  for(i=0;i<MAX_PLAINTEXT_SIZE;i++){
   Serial.printf("%02X",output[i]);
  }


  Serial.println();

  Serial.println("Decrypted AES message:");
  for(i=0;i<MAX_PLAINTEXT_SIZE;i++){
   Serial.printf("%c",output2[i]);
  }


  Serial.println();
  Serial.println();

}



void setup()
{

  // initialize LED digital pin as an output.
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
  //while (!Serial.available()) {}

  delay(10000);

    //   Serial.println();
    //   Serial.println();
    //   Serial.println(ssid);
    //   Serial.println(password);

  // Connect to WiFi network
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  //set static ip port
  //WiFi.config(IPAddress(192,168,10,118), IPAddress(192,168,10,1), IPAddress(255,255,255,0));

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(200);
  }
```

```
  Serial.print("Wifi Connected.! ESP IP is: ");
  Serial.println(WiFi.localIP());
  Serial.println();

  wifiServer.begin();
  Serial.printf("Server started, at %s \n", WiFi.localIP().toString().c_str());

  test_aes();
  test_saturnin();

}

void loop()
{

  WiFiClient client = wifiServer.available();

  if (client) {

    while (client.connected()) {

      while (client.available()>0) {
       char c = client.read();

       if(c=='0'){
          for (int i = 0; i< cipher_repeat; i++){
             memcpy(in,message,32);
          }
        array_to_string(message,32,myhexstring);
        client.write(myhexstring);
        //client.write("This is my top secret message..!");
        //Serial.println("Sending Unencrypted");
       }
       if(c=='1'){

        StartTime = millis();
        for (int i = 0; i< cipher_repeat; i++){
          memcpy(in,message,32);
          ctr.setKey(mykey, 32);
          ctr.setIV(myiv,16);
          ctr.setCounterSize(4);
          ctr.encrypt(output,in,MAX_PLAINTEXT_SIZE);
        }
        CurrentTime = millis();
        ElapsedTime = CurrentTime - StartTime;
        memcpy(in,message,32);
        ctr.setKey(mykey, 32);
        ctr.setIV(myiv,16);
        ctr.setCounterSize(4);
        ctr.encrypt(output,in,MAX_PLAINTEXT_SIZE);

        array_to_string(output,32,myhexstring);

        client.write(myhexstring);
        //Serial.println("Send AES encrypted");

        ltoa(ElapsedTime,mytimestr,10);
        Serial.print("AES time: ");
        Serial.println(mytimestr);
       }

       if(c=='2'){
          StartTime = millis();
```

```
        for (int i = 0; i< cipher_repeat; i++){
          // Saturnin key and original message
          memcpy(in,message,32);
          // Saturnin
          saturnin_block_encrypt(10,6,mykey,in);
        }
        CurrentTime = millis();
        ElapsedTime = CurrentTime - StartTime;
        memcpy(in,message,32);
        // Saturnin
        saturnin_block_encrypt(10,6,mykey,in);
        array_to_string(in,32,myhexstring);
        client.write(myhexstring);
        //Serial.println("Send SATURNIN encrypted");

        ltoa(ElapsedTime,mytimestr,10);
        Serial.print("SATURNIN time: ");
        Serial.println(mytimestr);
        // ltoa(StartTime,mytimestr,10);
        // Serial.print("StartTime: ");
        // Serial.println(mytimestr);
        // ltoa(CurrentTime,mytimestr,10);
        // Serial.print("CurrentTime: ");
        // Serial.println(mytimestr);
        Serial.println("");

      }

     }

     delay(10);
   }

   client.stop();

 }

}


/* ================================================================================ */
/*
 * Saturnin block cipher implementation (reference code, not optimized).
 */

#include <string.h>
#include <stdint.h>

/*
 * Compute round constants for R super-rounds and domain D.
 * Assumptions:
 *   0 <= R <= 31
 *   0 <= D <= 15
 */
static void
make_round_constants(int R, int D, uint16_t *RC0, uint16_t *RC1)
{
   uint16_t x0, x1;
   int n;

   x0 = x1 = D + (R << 4) + 0xFE00;

   for (n = 0; n < R; n ++) {
      int i;
```

```
    for (i = 0; i < 16; i ++) {
      x0 = (x0 << 1) ^ (0x2D & -(x0 >> 15));
      x1 = (x1 << 1) ^ (0x53 & -(x1 >> 15));
    }
    RC0[n] = x0;
    RC1[n] = x1;
  }
}

/*
 * Apply the S-boxes on the state (sigma_0 and sigma_1).
 */
static void
S_box(uint16_t *state)
{
  int i;

  for (i = 0; i < 16; i += 8) {
    uint16_t a, b, c, d;

    /* sigma_0 */
    a = state[i + 0];
    b = state[i + 1];
    c = state[i + 2];
    d = state[i + 3];
    a ^= b & c;
    b ^= a | d;
    d ^= b | c;
    c ^= b & d;
    b ^= a | c;
    a ^= b | d;
    state[i + 0] = b;
    state[i + 1] = c;
    state[i + 2] = d;
    state[i + 3] = a;

    /* sigma_1 */
    a = state[i + 4];
    b = state[i + 5];
    c = state[i + 6];
    d = state[i + 7];
    a ^= b & c;
    b ^= a | d;
    d ^= b | c;
    c ^= b & d;
    b ^= a | c;
    a ^= b | d;
    state[i + 4] = d;
    state[i + 5] = b;
    state[i + 6] = a;
    state[i + 7] = c;
  }
}

/*
 * Apply the inverse S-boxes on the state (inv_sigma_0 and inv_sigma_1).
 */
static void
S_box_inv(uint16_t *state)
{
  int i;

  for (i = 0; i < 16; i += 8) {
```

```
        uint16_t a, b, c, d;

        /* inv_sigma_0 */
        b = state[i + 0];
        c = state[i + 1];
        d = state[i + 2];
        a = state[i + 3];
        a ^= b | d;
        b ^= a | c;
        c ^= b & d;
        d ^= b | c;
        b ^= a | d;
        a ^= b & c;
        state[i + 0] = a;
        state[i + 1] = b;
        state[i + 2] = c;
        state[i + 3] = d;

        /* inv_sigma_1 */
        d = state[i + 4];
        b = state[i + 5];
        a = state[i + 6];
        c = state[i + 7];
        a ^= b | d;
        b ^= a | c;
        c ^= b & d;
        d ^= b | c;
        b ^= a | d;
        a ^= b & c;
        state[i + 4] = a;
        state[i + 5] = b;
        state[i + 6] = c;
        state[i + 7] = d;
    }
}

/*
 * Apply the linear transform (MDS) on the state.
 */
static void
MDS(uint16_t *state)
{
    uint16_t x0, x1, x2, x3, x4, x5, x6, x7;
    uint16_t x8, x9, xa, xb, xc, xd, xe, xf;

    x0 = state[0x0];
    x1 = state[0x1];
    x2 = state[0x2];
    x3 = state[0x3];
    x4 = state[0x4];
    x5 = state[0x5];
    x6 = state[0x6];
    x7 = state[0x7];
    x8 = state[0x8];
    x9 = state[0x9];
    xa = state[0xa];
    xb = state[0xb];
    xc = state[0xc];
    xd = state[0xd];
    xe = state[0xe];
    xf = state[0xf];

#define MUL(t0, t1, t2, t3)   do { \
        uint16_t mul_tmp = (t0); \
```

```
        (t0) = (t1); \
        (t1) = (t2); \
        (t2) = (t3); \
        (t3) = mul_tmp ^ (t0); \
    } while (0)

    x8 ^= xc; x9 ^= xd; xa ^= xe; xb ^= xf; /* C ^= D */
    x0 ^= x4; x1 ^= x5; x2 ^= x6; x3 ^= x7; /* A ^= B */
    MUL(x4, x5, x6, x7);                    /* B = MUL(B) */
    MUL(xc, xd, xe, xf);                    /* D = MUL(D) */
    x4 ^= x8; x5 ^= x9; x6 ^= xa; x7 ^= xb; /* B ^= C */
    xc ^= x0; xd ^= x1; xe ^= x2; xf ^= x3; /* D ^= A */
    MUL(x0, x1, x2, x3);                    /* A = MUL(A) */
    MUL(x0, x1, x2, x3);                    /* A = MUL(A) */
    MUL(x8, x9, xa, xb);                    /* C = MUL(C) */
    MUL(x8, x9, xa, xb);                    /* C = MUL(C) */
    x8 ^= xc; x9 ^= xd; xa ^= xe; xb ^= xf; /* C ^= D */
    x0 ^= x4; x1 ^= x5; x2 ^= x6; x3 ^= x7; /* A ^= B */
    x4 ^= x8; x5 ^= x9; x6 ^= xa; x7 ^= xb; /* B ^= C */
    xc ^= x0; xd ^= x1; xe ^= x2; xf ^= x3; /* D ^= A */

#undef MUL

    state[0x0] = x0;
    state[0x1] = x1;
    state[0x2] = x2;
    state[0x3] = x3;
    state[0x4] = x4;
    state[0x5] = x5;
    state[0x6] = x6;
    state[0x7] = x7;
    state[0x8] = x8;
    state[0x9] = x9;
    state[0xa] = xa;
    state[0xb] = xb;
    state[0xc] = xc;
    state[0xd] = xd;
    state[0xe] = xe;
    state[0xf] = xf;
}

/*
 * Apply the inverse of the linear transform (MDS) on the state.
 */
static void
MDS_inv(uint16_t *state)
{
    uint16_t x0, x1, x2, x3, x4, x5, x6, x7;
    uint16_t x8, x9, xa, xb, xc, xd, xe, xf;

    x0 = state[0x0];
    x1 = state[0x1];
    x2 = state[0x2];
    x3 = state[0x3];
    x4 = state[0x4];
    x5 = state[0x5];
    x6 = state[0x6];
    x7 = state[0x7];
    x8 = state[0x8];
    x9 = state[0x9];
    xa = state[0xa];
    xb = state[0xb];
    xc = state[0xc];
    xd = state[0xd];
```

```
    xe = state[0xe];
    xf = state[0xf];

#define MULinv(t0, t1, t2, t3)   do { \
        uint16_t mul_tmp = (t3); \
        (t3) = (t2); \
        (t2) = (t1); \
        (t1) = (t0); \
        (t0) = mul_tmp ^ (t1); \
    } while (0)

    x4 ^= x8; x5 ^= x9; x6 ^= xa; x7 ^= xb; /* B ^= C */
    xc ^= x0; xd ^= x1; xe ^= x2; xf ^= x3; /* D ^= A */
    x8 ^= xc; x9 ^= xd; xa ^= xe; xb ^= xf; /* C ^= D */
    x0 ^= x4; x1 ^= x5; x2 ^= x6; x3 ^= x7; /* A ^= B */
    MULinv(x0, x1, x2, x3);              /* A = MULinv(A) */
    MULinv(x0, x1, x2, x3);              /* A = MULinv(A) */
    MULinv(x8, x9, xa, xb);              /* C = MULinv(C) */
    MULinv(x8, x9, xa, xb);              /* C = MULinv(C) */
    x4 ^= x8; x5 ^= x9; x6 ^= xa; x7 ^= xb; /* B ^= C */
    xc ^= x0; xd ^= x1; xe ^= x2; xf ^= x3; /* D ^= A */
    MULinv(x4, x5, x6, x7);              /* B = MULinv(B) */
    MULinv(xc, xd, xe, xf);             /* D = MULinv(D) */
    x8 ^= xc; x9 ^= xd; xa ^= xe; xb ^= xf; /* C ^= D */
    x0 ^= x4; x1 ^= x5; x2 ^= x6; x3 ^= x7; /* A ^= B */

#undef MULinv

    state[0x0] = x0;
    state[0x1] = x1;
    state[0x2] = x2;
    state[0x3] = x3;
    state[0x4] = x4;
    state[0x5] = x5;
    state[0x6] = x6;
    state[0x7] = x7;
    state[0x8] = x8;
    state[0x9] = x9;
    state[0xa] = xa;
    state[0xb] = xb;
    state[0xc] = xc;
    state[0xd] = xd;
    state[0xe] = xe;
    state[0xf] = xf;
}

/*
 * Apply the SR_slice permutation.
 */
static void
SR_slice(uint16_t *state)
{
    int i;

    for (i = 0; i < 4; i ++) {
        state[ 4 + i] = ((state[ 4 + i] & 0x7777) << 1)
            | ((state[ 4 + i] & 0x8888) >> 3);
        state[ 8 + i] = ((state[ 8 + i] & 0x3333) << 2)
            | ((state[ 8 + i] & 0xcccc) >> 2);
        state[12 + i] = ((state[12 + i] & 0x1111) << 3)
            | ((state[12 + i] & 0xeeee) >> 1);
    }
}
```

```
/*
 * Apply the inverse of the SR_slice permutation.
 */
static void
SR_slice_inv(uint16_t *state)
{
   int i;

   for (i = 0; i < 4; i ++) {
      state[ 4 + i] = ((state[ 4 + i] & 0x1111) << 3)
         | ((state[ 4 + i] & 0xeeee) >> 1);
      state[ 8 + i] = ((state[ 8 + i] & 0x3333) << 2)
         | ((state[ 8 + i] & 0xcccc) >> 2);
      state[12 + i] = ((state[12 + i] & 0x7777) << 1)
         | ((state[12 + i] & 0x8888) >> 3);
   }
}

/*
 * Apply the SR_sheet permutation.
 */
static void
SR_sheet(uint16_t *state)
{
   int i;

   for (i = 0; i < 4; i ++) {
      state[ 4 + i] = ((state[ 4 + i] <<  4) | (state[ 4 + i] >> 12));
      state[ 8 + i] = ((state[ 8 + i] <<  8) | (state[ 8 + i] >>  8));
      state[12 + i] = ((state[12 + i] << 12) | (state[12 + i] >>  4));
   }
}

/*
 * Apply the inverse of the SR_sheet permutation.
 */
static void
SR_sheet_inv(uint16_t *state)
{
   int i;

   for (i = 0; i < 4; i ++) {
      state[ 4 + i] = ((state[ 4 + i] << 12) | (state[ 4 + i] >>  4));
      state[ 8 + i] = ((state[ 8 + i] <<  8) | (state[ 8 + i] >>  8));
      state[12 + i] = ((state[12 + i] <<  4) | (state[12 + i] >> 12));
   }
}

/*
 * XOR the key into the state.
 */
static void
XOR_key(const uint16_t *key, uint16_t *state)
{
   int i;

   for (i = 0; i < 16; i ++) {
      state[i] ^= key[i];
   }
}

/*
 * XOR the rotated key into the state.
 */
```

```
static void
XOR_key_rotated(const uint16_t *key, uint16_t *state)
{
   int i;

   for (i = 0; i < 16; i ++) {
      state[i] ^= (key[i] << 11) | (key[i] >> 5);
   }
}

/*
 * Perform one Saturnin block encryption.
 *   R     number of super-rounds (0 to 31)
 *   D     separation domain (0 to 15)
 *   key   key (32 bytes)
 *   buf   block to encrypt
 * The 'key' and 'buf' buffers may overlap. The encrypted block is
 * written back in 'buf'.
 */
void
saturnin_block_encrypt(int R, int D, const uint8_t *key, uint8_t *buf)
{
   uint16_t RC0[31], RC1[31];
   uint16_t xk[16], xb[16];
   int i;

   /*
    * Decode key and input block.
    */
   for (i = 0; i < 16; i ++) {
      xk[i] = key[i << 1] + ((uint16_t)key[(i << 1) + 1] << 8);
      xb[i] = buf[i << 1] + ((uint16_t)buf[(i << 1) + 1] << 8);
   }

   /*
    * Compute round constants.
    */
   make_round_constants(R, D, RC0, RC1);

   /*
    * XOR key into state.
    */
   XOR_key(xk, xb);

   /*
    * Run all rounds (two rounds per super-round).
    */
   for (i = 0; i < R; i ++) {
      /*
       * Even round.
       */
      S_box(xb);
      MDS(xb);

      /*
       * Odd round.
       */
      S_box(xb);
      if ((i & 1) == 0) {
         /*
          * Round r = 1 mod 4.
          */
         SR_slice(xb);
         MDS(xb);
```

```
            SR_slice_inv(xb);
            xb[0] ^= RC0[i];
            xb[8] ^= RC1[i];
            XOR_key_rotated(xk, xb);
        } else {
            /*
             * Round r = 3 mod 4.
             */
            SR_sheet(xb);
            MDS(xb);
            SR_sheet_inv(xb);
            xb[0] ^= RC0[i];
            xb[8] ^= RC1[i];
            XOR_key(xk, xb);
        }
    }

    /*
     * Encode output block.
     */
    for (i = 0; i < 16; i ++) {
        buf[(i << 1) + 0] = (uint8_t)xb[i];
        buf[(i << 1) + 1] = (uint8_t)(xb[i] >> 8);
    }
}

/*
 * Perform one Saturnin block decryption.
 *   R     number of super-rounds (0 to 31)
 *   D     separation domain (0 to 15)
 *   key   key (32 bytes)
 *   buf   block to decrypt
 * The 'key' and 'buf' buffers may overlap. The decrypted block is
 * written back in 'buf'.
 */
void
saturnin_block_decrypt(int R, int D, const uint8_t *key, uint8_t *buf)
{
    uint16_t RC0[31], RC1[31];
    uint16_t xk[16], xb[16];
    int i;

    /*
     * Decode key and input block.
     */
    for (i = 0; i < 16; i ++) {
        xk[i] = key[i << 1] + ((uint16_t)key[(i << 1) + 1] << 8);
        xb[i] = buf[i << 1] + ((uint16_t)buf[(i << 1) + 1] << 8);
    }

    /*
     * Compute round constants.
     */
    make_round_constants(R, D, RC0, RC1);

    /*
     * Run all rounds (two rounds per super-round).
     */
    for (i = R - 1; i >= 0; i --) {
        /*
         * Odd round.
         */
        if ((i & 1) == 0) {
            /*
```

```
            * Round r = 1 mod 4.
            */
            XOR_key_rotated(xk, xb);
            xb[0] ^= RC0[i];
            xb[8] ^= RC1[i];
            SR_slice(xb);
            MDS_inv(xb);
            SR_slice_inv(xb);
        } else {
            /*
            * Round r = 3 mod 4.
            */
            XOR_key(xk, xb);
            xb[0] ^= RC0[i];
            xb[8] ^= RC1[i];
            SR_sheet(xb);
            MDS_inv(xb);
            SR_sheet_inv(xb);
        }
        S_box_inv(xb);

        /*
        * Even round.
        */
        MDS_inv(xb);
        S_box_inv(xb);
    }

    /*
    * XOR key into state.
    */
    XOR_key(xk, xb);

    /*
    * Encode output block.
    */
    for (i = 0; i < 16; i ++) {
        buf[(i << 1) + 0] = (uint8_t)xb[i];
        buf[(i << 1) + 1] = (uint8_t)(xb[i] >> 8);
    }
}
```