Ανοιχτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακό Πρόγραμμα Σπουδών

Ασφάλεια Υπολογιστών και Δικτύων

Μεταπτυχιακή Διατριβή

Κρυπτογράφηση χαμηλών πόρων – Εφαρμογές στο πρωτόκολλο TLS

Ευμορφία Μαρία Μακρανδρέου

Επιβλέπων Καθηγητής

Δρ Κωνσταντίνος Λιμνιώτης

Μάιος 2023

Ανοιχτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακό Πρόγραμμα Σπουδών

Ασφάλεια Υπολογιστών και Δικτύων

Μεταπτυχιακή Διατριβή

Κρυπτογράφηση χαμηλών πόρων – Εφαρμογές στο πρωτόκολλο TLS

Ευμορφία Μαρία Μακρανδρέου

Επιβλέπων Καθηγητής

Δρ 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 2023

ΛΕΥΚΗ ΣΕΛΙΔΑ

Περιεχόμενα

Περίληψη

Το Transport Layer Security (TLS) 1.3 πρωτόκολλο είναι από τα πιο διαδεδομένα πρωτόκολλα, το οποίο έχει ως σκοπό την ασφαλή μεταφορά δεδομένων μεταξύ client-server. Οι αλγόριθμοι κρυπτογράφησης που ήδη χρησιμοποιούνται στο πρωτόκολλο TLS, λόγω της αυξημένης ισχύς που καταναλώνουν, δε δύνανται να χρησιμοποιηθούν σε συσκευές περιορισμένων πόρων. Το NIST (National Institute of Standards and Technology) προέβη στην διεξαγωγή διαγωνισμού, με σκοπό την ανάδειξη ενός πρότυπου lightweight αλγορίθμου. Στο στο τρίτο και τελικό γύρο της διαδικασίας, από τους δέκα που επικράτησαν, ο ASCON με δύο παραλλαγές, αναδείχθηκε ως ο πλέον κατάλληλος για προτυποποίηση. Η παρούσα διατριβή επικεντρώνεται στην ανάλυση της κρυπτογράφησης χαμηλών πόρων με αναφορά σε βασικές έννοιες της, όπως η ασύμμετρη και συμμετρική, όπου καθεμία εξ αυτών παρουσιάζει ποικίλα χαρακτηριστικά. Εν συνεχεία, παρουσιάζεται το πρωτόκολλο TLS και γίνεται αναφορά και σύγκριση των δύο τελευταίων εκδόσεών του, καθώς και των σουιτών κρυπτογράφησης του πρωτοκόλλου. Έπειτα, ακολουθεί περιγραφή, ανάλυση και σύγκριση των δέκα αλγορίθμων χαμηλών πόρων μέσω πειραμάτων με σκοπό την ανάδειξη του επικρατέστερου-κατάλληλου για την μελλοντική προσθήκη του στο πρωτόκολλο TLS 1.3. Τέλος ακολουθούν πιο αναλυτικά τα πειράματα και οι μετρήσεις που πήραμε, συνοδευόμενα από μια ανάλυση των δεδομένων του κάθε αλγορίθμου ξεχωριστά με την ανάδειξη του πιο αξιόπιστου από πλευρά αποδοτικότητας.

Summary

The Transport Layer Security (TLS) 1.3 protocol is one of the most widely used protocols that aims to secure data transfer between client-server. The encryption algorithms that have already been used in the TLS protocol cannot be used on lightweight devices due to the increased power they consume. The NIST (National Institute of Standards and Technology) held a competition to find a standard lightweight algorithm, and in the third and final round of the process, ASCON with two variants emerged as the most suitable for standardization out of the ten finalists algorithms.

This thesis focuses on the analysis of low resource encryption with reference to its basic concepts such as asymmetric and symmetric, each of them presenting different characteristics. Subsequently, the TLS protocol is introduced and the latest two versions of the protocol are discussed and compared, as well as the protocol's encryption suites. Next, we describe, analyze and compare the ten lightweight algorithms through experiments in order to identify the most predominant-suitable one for future addition to the TLS 1.3 protocol. Finally, the experiments and the measurements obtained follow in more detail, accompanied by an analysis of the data of each algorithm separately.

Ευχαριστίες

Με την παρούσα διπλωματική εργασία ολοκληρώνονται οι σπουδές μου στο μεταπτυχιακό πρόγραμμα σπουδών με τίτλο « ΑΣΦΑΛΕΙΑ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΔΙΚΤΥΩΝ».

Κατά την διάρκεια των σπουδών μου ήταν καθοριστικός ο ρόλος των καθηγητών μου στα γνωστικά αντικείμενα που παρακολούθησα, στους οποίους και νιώθω την ανάγκη να εκφράσω ένα ειλικρινές ευχαριστώ για την πολύτιμη βοήθειά τους στην ολοκλήρωση των σπουδών μου.

Ιδιαίτερα επιθυμώ να ευχαριστήσω τον καθηγητή μου και επιβλέποντα της παρούσας διπλωματική εργασία, Κύριο Κωνσταντίνο Λιμνιώτη για την εμπιστοσύνη που μου έδειξε από την αρχή, αναθέτοντάς μου το συγκεκριμένο θέμα, την επιστημονική του καθοδήγηση, τις υποδείξεις του, την υπομονή του και επιμονή του, το αμείωτο ενδιαφέρον του και την αδιάλειπτη υποστήριξη του σε όλα τα στάδια εκπόνησης της εργασίας με τις εύστοχες και πολύ εποικοδομητικές παρατηρήσεις του.

Τέλος, θα ήθελα εκφράσω την ευγνωμοσύνη μου στην οικογένειά μου για την συμπαράστασή της, τα υποστηρικτικά της λόγια και την κατανόησή της, καθ' όλη τη διάρκεια των σπουδών μου.

Κεφάλαιο 1

Εισαγωγή

Σήμερα όλο και περισσότεροι τομείς, (π.χ Διαδίκτυο των πραγμάτων - IoT, δίκτυα αισθητήρων, υγειονομική περίθαλψη κ.α.) οι οποίοι χρησιμοποιούν καθημερινά συσκευές με πολλούς περιορισμούς, αλληλοσυνδέονται μεταξύ τους, κυρίως ασύρματα, έτσι ώστε να επιτύχουν κάποια εργασία. Αν και η χρήση των συσκευών περιορισμένων λειτουργιών αυξάνεται ραγδαία, εξακολουθούν να αποτελούν ερευνητικές προκλήσεις, ζητήματα, που άπτονται τόσο της ασφάλειας όσο του σχεδιασμού τους. Ένα από τα πιο ευρέως διαδεδομένα πρωτόκολλα ασφαλείας, τα οποία χρησιμοποιούνται για τη μεταφορά πληροφοριών ανάμεσα σε client/server, είναι το TLS, όπου για τη διασφάλιση των πληροφοριών χρησιμοποιεί αλγόριθμους κρυπτογράφησης, οι οποίοι όμως είναι σχεδιασμένοι για περιβάλλοντα desktop/server και ως εκ τούτου δεν προσφέρονται για τις ανωτέρω εφαρμογές που εμφανίζουν περιορισμούς. Ένας από αυτούς τους αλγορίθμους και ο πιο επικρατέστερος σήμερα, είναι το πρότυπο κρυπτογράφησης AES, το οποίο είναι μαζικής κρυπτογράφησης, ενσωματωμένο στη σουίτα κρυπτογράφησης του πρωτοκόλλου TLS και ως κύριο τρόπο λειτουργίας έχει τον Galois counter mode με κλειδί μεγέθους 128 bit ή 256 bit. Λόγω των ανωτέρω, αρχίζει να υπάρχει αύξηση στη ζήτηση lightweight κρυπτογραφικών προτύπων, που να είναι κατάλληλα, ώστε να διασφαλιστούν οι συσκευές αυτές, με τον οργανισμό NIST (National Institute of Standards and Technology) να έχει ήδη ξεκινήσει έρευνα για αυτό το κομμάτι. Μετά από μια διαδικασία αξιολόγησης αλγορίθμων χαμηλών πόρων και αφού τέθηκαν υπό εξέταση δέκα υποψήφιοι αλγόριθμοι προς τυποποίηση, ο αλγόριθμος ASCON αναδείχθηκε ως ο επικρατέστερος μεταξύ των υπολοίπων.

1.1 Ερευνητικά Ερωτήματα

- -Ποιος από τους επικρατέστερους lightweight αλγόριθμους έχει την καλύτερη απόδοση, σε λειτουργία ενός πρωτοκόλλου ασφαλείας όπως το TLS που βασίζεται σε παραδοσιακούς κρυπτογραφικούς αλγορίθμους;

- -Θα μπορέσει ένας ή και παραπάνω από αυτούς τους 10 αλγόριθμους να φτάσει μία ικανοποιητική απόδοση σε διάφορα περιβάλλοντα με περιορισμούς ώστε να δύναται να ενσωματωθεί απευθείας στο πρωτόκολλο ασφαλείας TLS;
- -Πόσο ασφαλής θα είναι στη πραγματικότητα-πρακτικά η μεταφορά πληροφοριών στις συσκευές περιορισμένων λειτουργιών μετά την αντικατάσταση του αλγορίθμου κρυπτογράφησης και σε τί ποσοστό;

1.2 Αντικείμενο

Η εργασία αυτή εστιάζει στην ελαφριά κρυπτογραφία και συγκεκριμένα στους δέκα lightweight αλγορίθμους που έφτασαν μέχρι το τρίτο και τελικό στάδιο του διαγωνισμού του Εθνικού Ινστιτούτου Προτύπων και Τεχνολογίας (NIST), οι οποίοι υπόκεινται σε συγκρίσεις με σκοπό την ανάδειξη του πιο αξιόπιστου, ως ο καταλληλότερος με την δυνατότητα εφαρμογής στο πρωτόκολλο TLS.

1.3 Μεθοδολογία

Λήψη πληροφοριακού υλικού από ερευνητικές διαδικτυακές ιστοσελίδες με σκοπό τον εμπλουτισμό της θεωρητικής μας ανάλυσης.

Ανάλυση των δέκα φιναλίστ αλγορίθμων σε συσκευή περιορισμένων πόρων και φυσικού περιβάλλοντος, λαμβάνοντας υπόψιν την αποδοτικότητά τους για να εξεταστεί εν συνεχεία συγκριτικά.

Λήψη πληροφοριών που αφορούν την απόδοση των αλγορίθμων μέσω διεξαγωγής πειραμάτων σε συσκευή χαμηλών πόρων.

1.4 Δομή Μεταπτυχιακής διατριβής

Η δομή της παρούσας διατριβής είναι η παρακάτω:

Στο κεφάλαιο 1 γίνεται μια εισαγωγή στο αντικείμενο της εργασίας μας και παρουσιάζονται τα ερευνητικά ερωτήματα, τα οποία αποτέλεσαν το ερέθισμα της θεωρητικής και πρακτικής μας ανάλυσης.

Στο κεφάλαιο 2 παρουσιάζονται οι βασικές έννοιες κρυπτογράφησης χαμηλών πόρων καθώς και οι δύο διαφορετικές κατηγορίες, συνοδευόμενες από τα χαρακτηριστικά που διαφοροποιούν τη μία από την άλλη.

Στο κεφάλαιο 3 γίνεται εισαγωγή στο πρωτόκολλο TLS, περιγραφή, ανάλυση και σύγκριση των εκδόσεων 1.2 και 1.3 καθώς και ανάλυση της σουίτας κρυπτογράφησης.

Το κεφάλαιο 4 αφιερώνεται στην περιγραφή και στην εκτεταμένη ανάλυση των δέκα επικρατέστερων αλγορίθμων χαμηλών πόρων, με αναφορές στα χαρακτηριστικά τους και τον τρόπο λειτουργίας τους.

Στο κεφάλαιο 5 γίνεται παρουσίαση των πειραμάτων που διεξήχθησαν, οι μετρήσεις που πάρθηκαν μαζί με τα δεδομένα του κάθε αλγορίθμου χαμηλών πόρων και τέλος συγκριτική αξιολόγηση των αλγορίθμων, ως προς την απόδοση τους.

Στο κεφάλαιο 6 γίνεται μια ανασκόπηση της εργασίας μας, η οποία περιλαμβάνει τα συμπεράσματά μας ως προς τις αναλύσεις που διεξήχθησαν, συνοδευόμενη από την βιβλιογραφία με τις πηγές της, οι οποίες μας βοήθησαν να εμπλουτίσουμε την θεωρητική και πρακτική μας ανάλυση.

Κεφάλαιο 2

Κρυπτογράφηση χαμηλών πόρων

2.1 Βασικές έννοιες

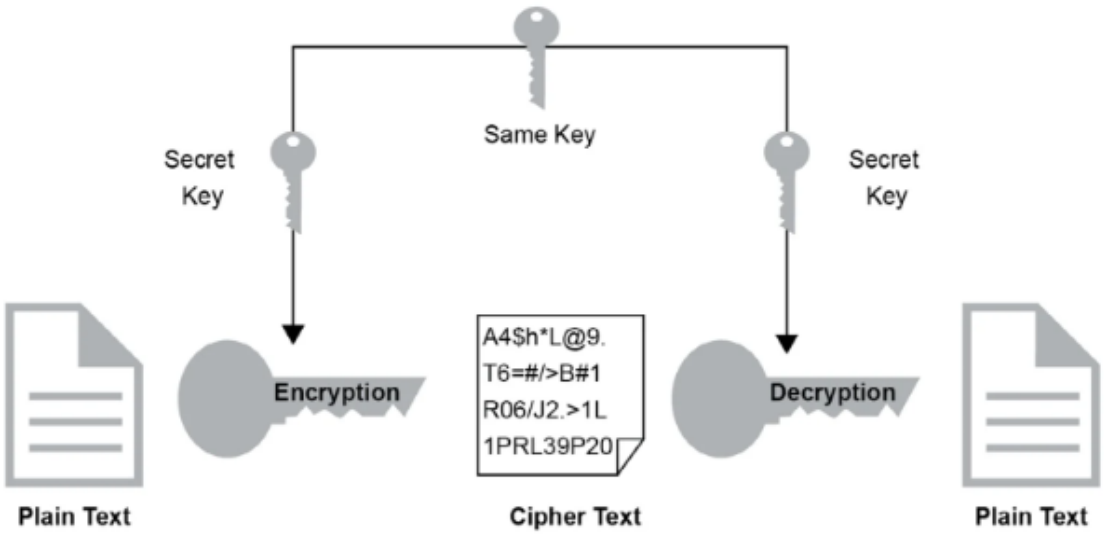
Η κρυπτογραφία χαμηλών πόρων στοχεύει σε συσκευές περιορισμένων πόρων, όπως RFID (Radio Frequency Identification), όπου μπορεί να εφαρμοστεί τόσο σε υλικό όσο και σε λογισμικό με διαφορετικές τεχνολογίες επικοινωνίας. Σε ένα περιβάλλον με περιορισμένους πόρους είναι δύσκολο να εφαρμοστούν οι τυπικοί κρυπτογραφικοί αλγόριθμοι λόγω του μεγέθους υλοποίησης, της ταχύτητας ή της απόδοσης και της κατανάλωσης ενέργειας. Η ελαφριά κρυπτογραφία αντισταθμίζει το κόστος υλοποίησης, την ταχύτητα, την απόδοση, την ασφάλεια και τη κατανάλωση ενέργειας σε συσκευές περιορισμένων πόρων.

Σκοπός της ελαφριάς κρυπτογράφησης είναι να χρησιμοποιεί όσο το δυνατόν λιγότερη μνήμη, λιγότερους υπολογιστικούς πόρους και τροφοδοτικό για να παρέχει λύσεις ασφαλείας που μπορούν να λειτουργήσουν σε συσκευές περιορισμένων πόρων (lightweight). Το μειονέκτημα της κρυπτογράφησης χαμηλών πόρων είναι ότι είναι λιγότερο ασφαλής διότι δεν εκμεταλλεύεται πάντα τους συμβιβασμούς ασφάλειας-αποδοτικότητας.

2.2 Συμμετρική Κρυπτογράφηση

Η συμμετρική κρυπτογράφηση ή αλλιώς κρυπτογραφία μυστικού κλειδιού, είναι η κρυπτογράφηση που γίνεται με τη χρήση ενός μυστικού κλειδιού για την κοινή χρήση κρυπτογραφημένων δεδομένων μεταξύ δύο μερών.

Ciphers αυτής της κατηγορίας ονομάζονται συμμετρικοί διότι χρησιμοποιούν το ίδιο κλειδί για τη κρυπτογράφηση και την αποκρυπτογράφηση των δεδομένων. Οι αλγόριθμοι αυτοί είναι καλύτεροι για μαζική κρυπτογράφηση διότι έχουν μικρότερο μέγεθος κλειδιού, όπου αυτό σημαίνει λιγότερο χώρο αποθήκευσης και ταχύτερη μετάδοση.

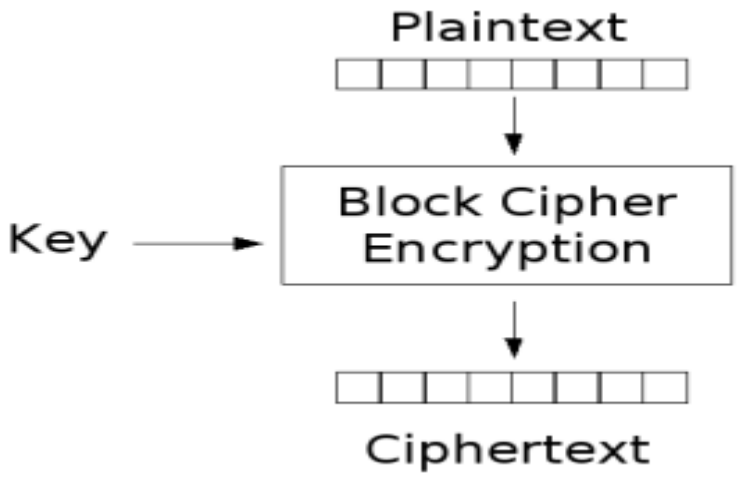


Σχήμα 2.1: *Συμμετρική κρυπτογράφηση*

2.2.1 Κρυπταλγόριθμος Τμήματος (Block Cipher)

Οι block ciphers είναι περισσότερο διαδεδομένοι, όπου το αρχικό το αρχικό μήνυμα χωρίζεται σε τμήματα σταθερού μεγέθους και το κάθε τμήμα κρυπτογραφείται ξεχωριστά.

Είναι σχεδιασμένοι για μεγάλα μπλοκ δεδομένων και έχουν μεγέθη μπλοκ που απαιτούν padding και χρησιμοποιούν ένα σταθερό αμετάβλητο μετασχηματισμό.

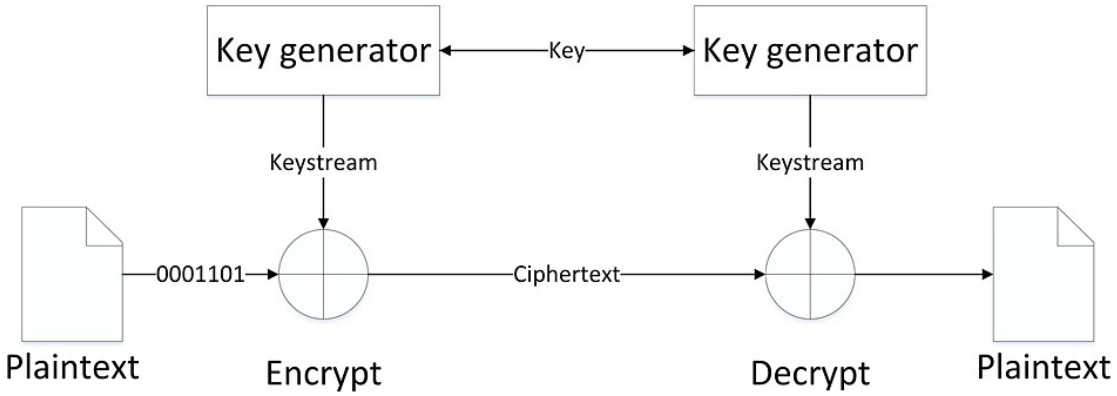


Σχήμα 2.2: *Κρυπτογράφηση μπλοκ*

2.2.2 Κρυπταλγόριθμος Ροής (Stream Cipher)

Ένας stream cipher είναι συμμετρικού κλειδιού όπου τα ψηφία απλού κειμένου συνδυάζονται με μία ψευδοτυχαία ροή ψηφίων κρυπτογράφησης (keystream). Κάθε ψηφίο απλού κειμένου κρυπτογραφείται ένα κάθε φορά με το αντίστοιχο ψηφίο της κλειδοροής, για να δώσει ένα ψηφίο του κρυπτοκειμένου ροής. Δεδομένου ότι η κρυπτογράφηση κάθε ψηφίου εξαρτάται από την τρέχουσα κατάσταση του cipher, είναι επίσης γνωστή ως κρυπτογράφηση κατάστασης (state cipher). Πρακτικά ένα ψηφίο είναι συνήθως ένα bit και η λειτουργία συνδυασμού είναι XOR.

Οι streams ciphers χρησιμοποιούνται σε εφαρμογές που απαιτείται υψηλή ταχύτητα και χαμηλή κατανάλωση ισχύος. Λειτουργούν καλά για μεγάλα ή μικρά κομμάτια δεδομένων αλλά είναι κατάλληλα για μικρότερα μεγέθη δεδομένων επειδή δεν απαιτείται μέγεθος μπλοκ.



Σχήμα 2.3: *Κρυπτογράφηση ροής*

2.2.3 Επαληθευμένη Κρυπτογράφηση με Συσχετισμένα δεδομένα (AEAD)

Η κρυπτογράφηση AEAD παρέχει ταυτόχρονα εμπιστευτικότητα μέσω κρυπτογράφησης και ακεραιότητα μέσω του ελέγχου ταυτότητας μηνυμάτων. Η κρυπτογράφηση παίρνει το απλό κείμενο (plaintext), τα συσχετισμένα δεδομένα (AD), το nonce και το κλειδί (key) ως είσοδο και εξάγει το κρυπτογραφημένο κείμενο (ciphertext) και την ετικέτα ελέγχου ταυτότητας (tag). Τα συσχετισμένα δεδομένα ελέγχονται με απλό κείμενο και δεν είναι κρυπτογραφημένα για λόγους ακεραιότητας. Το Nonce είναι τυχαίος αριθμός μιας χρήσης που χρησιμοποιείται για την ενίσχυση της ασφάλειας της κρυπτογράφησης κάτω από το ίδιο κλειδί. Η διαδικασία αποκρυπτογράφησης λαμβάνει το κρυπτογραφημένο κείμενο, τα συσχετισμένα δεδομένα, το nonce, το κλειδί και την ετικέτα ελέγχου ταυτότητας ως είσοδο και εξάγει το απλό κείμενο εάν ο έλεγχος ταυτότητας είναι επιτυχής. Εάν ο έλεγχος ταυτότητας αποτύχει, το απλό κείμενο δεν εξάγεται.

Δύο τύποι λειτουργίας AEAD που επιλέγονται από το NIST είναι κρυπτογράφησης μπλοκ όπου κάνει χρήση μπλοκ (block cipher) και λειτουργίας που βασίζεται σε σφουγγάρι (sponge) και χρησιμοποιεί μετάθεση (permutation). Και οι δύο τύποι χρησιμοποιούν λειτουργία XOR και padding.

Η επαληθευμένη κρυπτογράφηση με συσχετιζόμενα δεδομένα (AD) έχει πέντε κύριες παραμέτρους: keys, block (rate), state, tag,nonce:

1. **Κλειδιά**

Το μέγεθος του κλειδιού είναι κρίσιμο για την ασφάλεια ενός cipher. Εάν το μέγεθος του κλειδιού είναι μικρό, ο συνολικός αριθμός των κλειδιών μειώνεται και καθίσταται δυνατή μια επίθεση ωμής βίας (brute-force attack). Στη περίπτωση αυτή θα πρέπει το μέγεθος του κλειδιού να είναι επαρκώς ανθεκτικό σε επιθέσεις ωμής βίας. Ο NIST απαιτεί το μέγεθος του κλειδιού να είναι 128bit ή >128bit.

2. **Μέγεθος Μπλοκ**

Η ασφάλεια και ο χρόνος επεξεργασίας επηρεάζεται από το μέγεθος του μπλοκ, όπου η τιμή αυτή επηρεάζει τον όγκο των δεδομένων που υποβάλλονται σε επεξεργασία με ασφάλεια όταν η κρυπτογράφηση εκτελείται με το ίδιο κλειδί. Το μέγεθος του μπλοκ αναφέρεται και στον αριθμό των δυαδικών ψηφίων απλού κειμένου και κρυπτοκειμένου που δημιουργούνται ανά κρυπτογράφηση μπλοκ ή μετάθεση. Επομένως, επηρεάζει στην αύξηση ή τη μείωση του χρόνο επεξεργασίας.

3. **Μέγεθος κατάστασης**

Το μέγεθος state υποδεικνύει το μέγεθος του block cipher ή της μετάθεσης που χρησιμοποιείται στην κρυπτογραφική διαδικασία. Αυτή η παράμετρος καθορίζει το μέγεθος των προσωρινών μεταβλητών κατά τη διαδικασία κρυπτογράφησης ή αποκρυπτογράφησης και έτσι αυξάνει ή μειώνει τη χρήση της μνήμης RAM για την αποθήκευση της ενδιάμεσης κατάστασης. Το μέγεθος της κατάστασης επηρεάζει και το χρόνο επεξεργασίας λόγω ότι σχετίζεται με τον όγκο υπολογισμού που απαιτείται για την ενημέρωση της κατάστασης.

4. **Μέγεθος ετικέτας**

Το μέγεθος της ετικέτας καθορίζει το μέγεθος της ετικέτας ελέγχου ταυτότητας. Αυτό το μέγεθος επηρεάζει τον χρόνο επεξεργασίας κατά τη δημιουργία της ετικέτας ελέγχου ταυτότητας. Σύμφωνα με το NIST, το μέγεθος της ετικέτας πρέπει να είναι 64bit ή >64bit.

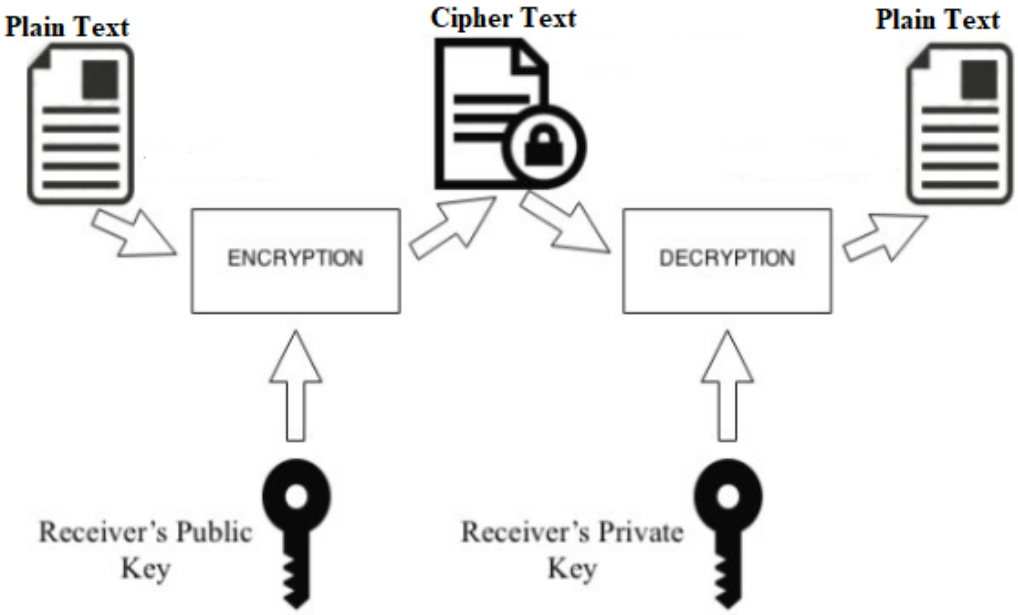
5. **Μέγεθος Nonce**

Το μέγεθος nonce καθορίζει το μέγεθος των τυχαίων αριθμών που χρησιμοποιούνται στην κρυπτογράφηση. Ο χρόνος που απαιτείται για τη δημιουργία και τη παροχή τυχαίων αριθμών εκτός της κρυπτογράφησης είναι ανάλογα με το μέγεθος των τυχαίων αριθμών που χρησιμοποιούνται για την κρυπτογράφηση. Επίσης ο χρόνος που απαιτείται για την επεξεργασία τυχαίων αριθμών στη κρυπτογράφηση διαφέρει, επηρεάζοντας έτσι το χρόνο επεξεργασίας. Επομένως, η παράμετρος αυτή επηρεάζει το χρόνο εκτέλεσης. Σύμφωνα με το NIST, το μέγεθος nonce πρέπει να είναι 96bit ή > 96bit.

2.3 Ασύμμετρη Κρυπτογράφηση

Η κρυπτογραφία δημόσιου κλειδιού ή ασύμμετρη κρυπτογράφηση είναι η κρυπτογράφηση που γίνεται με τη χρήση δύο κλειδιών. Αποτελείται από ένα δημόσιο κλειδί και ένα αντίστοιχο ιδιωτικό κλειδί . Τα ζεύγη κλειδιών δημιουργούνται με κρυπτογραφικούς αλγόριθμους που βασίζονται σε μαθηματικές πράξεις που ονομάζονται μονόδρομες συναρτήσεις .

Η ασφάλεια της κρυπτογράφησης δημόσιου κλειδιού εξαρτάται από την παραμονή της μυστικότητας του ιδιωτικού κλειδιού. Το δημόσιο κλειδί μπορεί να διανεμηθεί ανοιχτά χωρίς να υπάρχει πρόβλημα στην ασφάλεια του.



***Σχήμα 2.4**: Ασύμμετρη κρυπτογράφηση*

Κεφάλαιο 3

ΠρωτόκολλοTLS

3.1 Transport Layer Security (TLS)

Το TLS είναι ένα κρυπτογραφικό πρωτόκολλο που επιτρέπει στις εφαρμογές πελάτη/διακομιστή (client/server) να επικοινωνούν μέσω του διαδικτύου με ασφάλεια. Έχει σχεδιαστεί με τέτοιο τρόπο ώστε να αποτρέπει την υποκλοπή, την αλλοίωση και την πλαστογραφία μηνυμάτων. Αυτό επιτυγχάνεται αξιοποιώντας μία σειρά από πρωτόκολλα με τα σημαντικότερα να είναι το Handshake Protocol και το Record Protocol. Χρησιμοποιείται ευρέως σε εφαρμογές όπως κάποιες από αυτές να είναι το ηλεκτρονικό ταχυδρομείο στο πρωτόκολλο SMTP όπου SMTP + TLS (SMTPs), σε ιστοσελίδες στο πρωτόκολλο HTTP όπου HTTP + TLS (HTTPs) είναι και πιο ορατό και γνωστό στο ευρύ κοινό.

Οι πιο βασικές υπηρεσίες ασφάλειας που μας παρέχει το TLS είναι:

- **Αυθεντικοποίηση:** Το TLS κάνει επαλήθευση ταυτότητας του client/serverμε τη χρήση ασύμμετρης κρυπτογραφίας.
- **Εμπιστευτικότητα:** προστατεύει τα δεδομένα που ανταλλάσσονται, από μη εξουσιοδοτημένη πρόσβαση με τη χρήση συμμετρικών αλγορίθμων κρυπτογράφησης.
- **Ακεραιότητα:** αναγνωρίζει οποιαδήποτε αλλαγή στα δεδομένα κατά τη μετάδοση ελέγχοντας τον κωδικό ελέγχου ταυτότητας μηνύματος(Message Authentication Code).

Ένα παράδειγμα “https”ασφαλής ιστοσελίδας όπως φαίνεται παρακάτω:

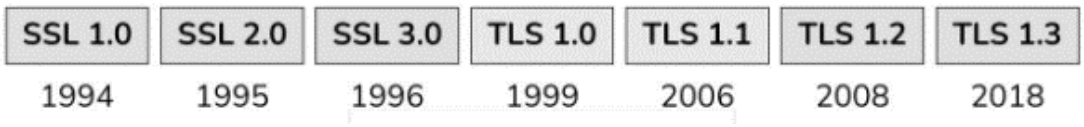
Εικόνα 3.1: *Παράδειγμα https σε ιστοσελίδα*

Ένα παράδειγμα “SMTPs” σε διακομιστή όπως φαίνεται παρακάτω:

Εικόνα 3.2: *Παράδειγμα SMTPs*

Το TLS κυκλοφόρησε αρχικά με την ονομασία SSL (Secure Socket Layer) από την Netscape communications το 1995και στη συνέχεια μετονομάστηκε με την τρέχουσα ονομασία από την IETF (Internet Engineering Task Force) το 1999.Κατά τη διάρκεια των χρόνων ο μεγάλος και αυξανόμενος αριθμός των ευπαθειών οδήγησε στην εφαρμογή διαδοχικών επεκτάσεων και βελτιώσεων, με αποτέλεσμα την δημιουργία ενός ασφαλέστερου και πιο αξιόπιστου πρωτοκόλλου. Το τελευταίο πραγματοποιήθηκε, φυσικά, μέσα από μια σειρά αναλύσεων και προσπαθειών. Η τρέχουσα έκδοση είναι το TLS 1.3, η οποία ορίστηκε τον Αύγουστο του 2018.Οι κοινοποιήσεις του TLSμέσω των RFCείναι οι ακόλουθες:

- **TLS 1.0→**RFC2246 το 1999.
- **TLS 1.1→**RFC4346 το 2006.
- **TLS 1.2→**RFC5246 το 2008.
- **TLS 1.3→**RFC8446 το 2018.



Εικόνα 3.3: *Εκδόσεις του TLS διαχρονικά*

3.2 Περιγραφή των TLS 1.2 και TLS 1.3

Το TLS αποτελεί ένα σύστημα κρυπτογράφησης υβριδικού τύπου, το οποίο κάνει χρήση συμμετρικής και ασύμμετρης κρυπτογράφησης με την δεύτερη να υστερεί στο χρόνο εφαρμογής λόγω του πιο αργού ρυθμού της. Η κρυπτογράφηση δημοσίου κλειδιού που πραγματοποιεί το TLS επιτρέπει στονclientκαι στον server να μοιράζονται με ασφαλή τρόπο ένα συμμετρικό κλειδί, το οποίο δύνανται στην συνέχεια να χρησιμοποιηθεί για την κρυπτογράφηση δια μέσου μη ασύμμετρου αλγορίθμου

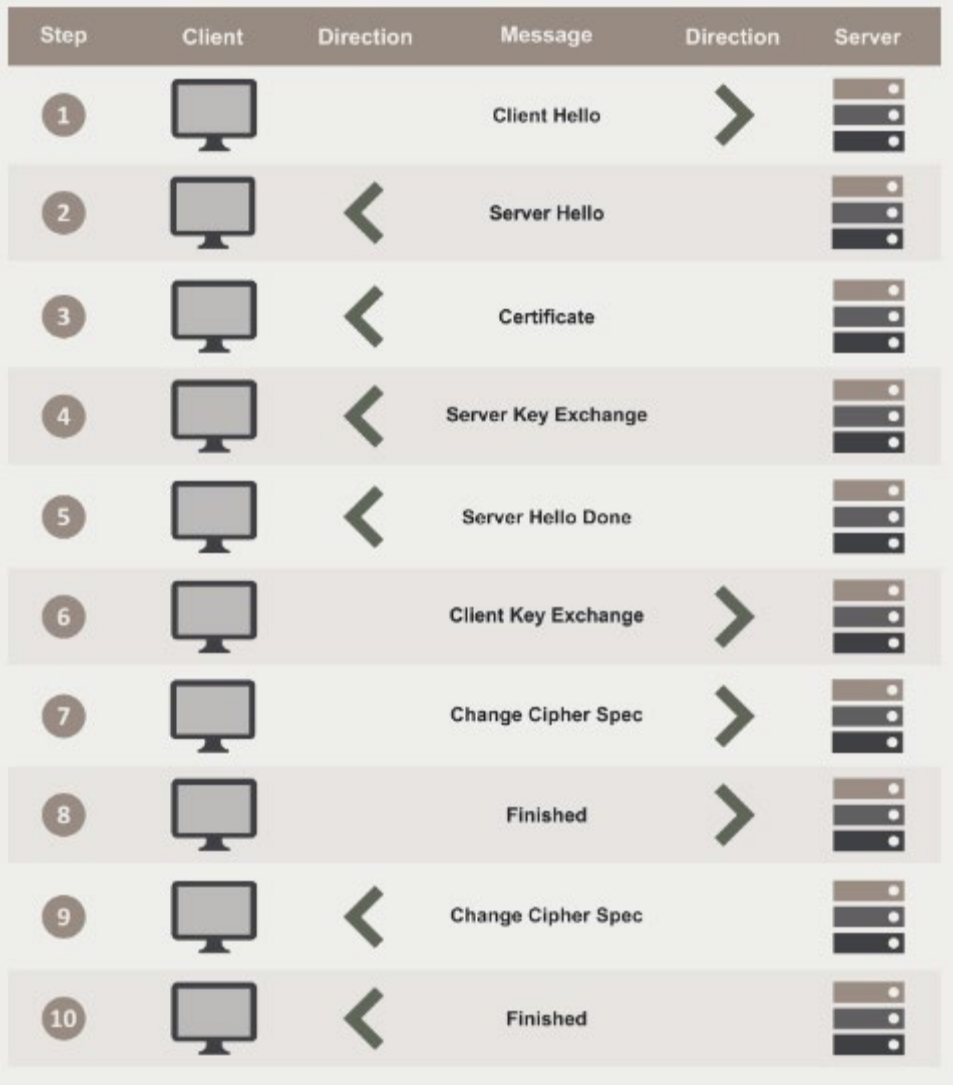
κρυπτογράφησης, καθ’ όλη τη διάρκεια της επικοινωνίας. Για να μπορέσει να διεξαχθεί η σύνδεση μεταξύ clientκαι serverκρίνεται απαραίτητη η συμφωνία ανάμεσά τους, για τις παραμέτρους, στις οποίες θα στηριχθεί η εν λόγω επικοινωνία. Η διαπραγμάτευση αυτή γίνεται μέσω του πρωτοκόλλου χειραψίας (handshake).

3.2.1 TLS 1.2 Handshake

Η χειραψία στο πρωτόκολλο TLS 1.2 είναι παρόμοια με αυτή των εκδόσεων 1.0 και 1.1.Παρακάτω περιγράφονται τα βήματα της διαδικασίας του handshake:

1. Η χειραψία ξεκινάει από τον πελάτη με την αποστολή ενός μηνύματος “Client Hello” στον διακομιστή. Το μήνυμα αποτελείται από κρυπτογραφικές πληροφορίες, όπως τα υποστηριζόμενα πρωτόκολλα και Cipher Suites αλλά και επίσης από κάποιες τυχαίες τιμές (random values).
2. Ως απάντηση ο διακομιστής απαντά με το μήνυμα "server hello". Αυτό το μήνυμα περιλαμβάνει το Cipher Suite που έχει επιλέξει ο διακομιστής από τα προσφερόμενα από τον πελάτη, το ψηφιακό πιστοποιητικό του, το αναγνωριστικό συνόδου (session ID) και κάποιες τυχαίες τιμές.
3. Ο πελάτης επαληθεύει το πιστοποιητικό που αποστέλλει ο διακομιστής και στέλνει πίσω μια τυχαία συμβολοσειρά bytes γνωστά ως "pre-master secret",τα οποία κρυπτογραφεί χρησιμοποιώντας το δημόσιο κλειδί του πιστοποιητικού του διακομιστή.
4. Μόλις ο διακομιστής λάβει το pre-master secret, ο πελάτης και ο διακομιστής δημιουργούν ένα κύριο κλειδί μαζί με τα κλειδιά συνόδου (εφήμερα κλειδιά) τα οποία θα χρησιμοποιηθούν για τη συμμετρική κρυπτογράφηση των δεδομένων.
5. Ένα μήνυμα "Change Cipher Spec" στέλνεται από τον πελάτη στον διακομιστή για να τον ενημερώσει ότι θα κάνει χρήση συμμετρικής κρυπτογράφησης με τη βοήθεια των εφήμερων κλειδιών και στέλνει το μήνυμα "Client Finished".
6. Τέλος ο διακομιστής απαντάει με μήνυμα "Change Cipher Spec" και αλλάζει τη κατάσταση ασφαλείας του σε συμμετρική κρυπτογράφηση. Επίσης ολοκληρώνει τη χειραψία με την αποστολή μηνύματος "server finished".

Για την ολοκλήρωση της χειραψίας παρατηρούμε ότι απαιτούνται 2 roundtrips μεταξύ client και server.



Εικόνα 3.4: *Παράδειγμα χειραψίας στο TLS 1.2*

3.2.2 TLS 1.3 Handshake

Στο πρωτόκολλο TLS 1.3 η διαδικασία της χειραψίας απαιτεί μόνο ένα roundtrip, κάτι το οποίο μειώνει αρκετά τη διάρκεια ολοκλήρωσης της. Ακολουθούν τα βήματα της διαδικασίας:

1. Η χειραψία ξεκινά στέλνοντας το μήνυμα "Client Hello" όπου μαζί στέλνει και μία λίστα με τα υποστηριζόμενα cipher suiteskαι μαντεύει ποιο πρωτόκολλο ανταλλαγής κλειδιού κατά πάσα πιθανότητα θα διαλέξει ο διακομιστής. Επίσης ο clientστέλνει και το δημόσιο κλειδί για το συγκεκριμένο πρωτόκολλο.
2. Ο διακομιστής απαντά στο "Client Hello"με το πρωτόκολλο ανταλλαγής κλειδιού που έχει επιλέξει. Το μήνυμα "Server Hello" περιλαμβάνει και το κοινόχρηστο κλειδί του διακομιστή, το πιστοποιητικό του καθώς και το μήνυμα "Server Finished".(Το μήνυμα "Server Finished", το οποίο εστάλη στο 6ο βήμα της χειραψίας TLS 1.2, στοTLS 1.3 αποστέλλεται δεύτερο βήμα.

Με αυτόν τον τρόπο, εξοικονομούνται τέσσερα βήματα και ένα roundtrip στην πορεία.).

3. Ο client ελέγχει το πιστοποιητικό του διακομιστή, δημιουργεί τα κλειδιά, χρησιμοποιώντας και τα δεδομένα του διακομιστή τα οποία θα χρησιμοποιήσει για την κρυπτογράφηση των δεδομένων. Τέλος στέλνει το μήνυμα "Client Finished" και αρχίζει η κρυπτογράφηση των δεδομένων.

Εικόνα 3.5: *Παράδειγμα χειραψίας στο TLS 1.3*

3.3 Σύγκριση TLS 1.2 με TLS 1.3

Η έκδοση 1.3 του πρωτόκολλου TLS έχει βελτιωθεί σημαντικά σε σχέση με την έκδοση 1.2 και ως προς την ασφάλεια αλλά και ως προς την απόδοση.

- Το TLSπλέον διαθέτει ασφαλέστερους μηχανισμούς ανταλλαγής κλειδιών στην έκδοση 1.3, όπου το ευάλωτο RSA και άλλες στατικές μέθοδοι ανταλλαγής κλειδιών που υπάρχουν στην 1.2, έχουν αφαιρεθεί, αφήνοντας μόνο το εφήμερο Diffie-Hellman ή το Elliptic-Curve Diffie-Hellman. Αυτό επιτυγχάνει τη τέλεια μυστικότητα προς τα εμπρός (forward secrecy).

- Στο TLS 1.3 η συμμετρική κρυπτογράφηση είναι πιο ασφαλής επειδή η χρήση AEAD είναι υποχρεωτική και αφαιρεί επίσης κάποιους αδύναμους αλγορίθμους από τη λίστα, όπως τον Block Cipher Mode (CBC), τον RC4 και τον Triple DES.

- Όπως αναφέραμε και πιο πάνω στη διαδικασία του handshake, το TLS 1.3 είναι τουλάχιστον 1 round-trip ταχύτερο από την έκδοση 1.2.

- Η σουίτα κρυπτογράφησης στο TLS 1.3 είναι απλούστερη, καθώς περιέχει τον αλγόριθμο AEAD και έναν αλγόριθμο κατακερματισμού. Οι αλγόριθμοι

ανταλλαγής κλειδιών και υπογραφής μεταφέρονται σε ξεχωριστά πεδία. Αντίθετα στο TLS 1.2, έχουν συγχωνευτεί στη σουίτα κρυπτογράφησης, όπου κάνει τον αριθμό των συνιστώμενων σουιτών κρυπτογράφησης να γίνεται πολύ μεγάλος, 37 επιλογές στο TLS 1.2 ενώ στο TLS 1.3 είναι μόνο 5.

- Στο TLS 1.3 η υπογραφή είναι ισχυρότερη, διότι υπογράφει ολόκληρη τη χειραψία, ενώ στο TLS 1.2 καλύπτει μόνο ένα μέρος της.

- Τέλος η κρυπτογραφία ελλειπτικών καμπυλών λαμβάνει σημαντική προσοχή στο TLS 1.3, με την προσθήκη καλύτερων αλγορίθμων καμπυλών, όπως ο αλγόριθμος ψηφιακής υπογραφής Edward-curve, ο οποίος είναι ταχύτερος χωρίς να θυσιάζει την ασφάλεια.

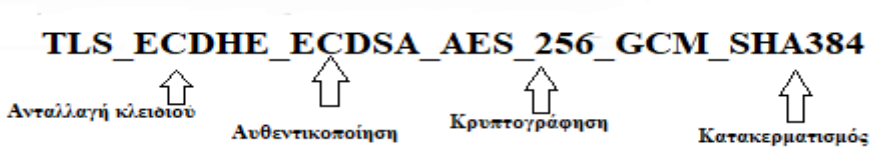
TLS 1.2	TLS 1.3
RSA, ECDH, ECDHE	ECDHE >> forward secrecy
AEAD, CBC, RC4, 3DES	AEAD
DHE_RSA_WITH_AES_256_CBC_SHA256	AES_256_GCM_SHA384
2-RTT	1-RTT, 0-RTT
Υπογράφει ένα κομμάτι της χειραψίας	Υπογράφει όλη τη χειραψία
ECDSA (P-256, P-384)	EdDSA (Ed25519, Ed448)

Πίνακας 3.6: *Βελτιώσεις του TLS 1.3 με το TLS 1.2*

3.4 Σουίτα κρυπτογράφησης

Όπως αναφέραμε και παραπάνω, η ασφάλεια σχεδόν κάθε σύνδεσης στο διαδίκτυο εξαρτάται από τη κρυπτογράφηση TLS. Το επίπεδο προστασίας για αυτές τις συνδέσεις καθορίζεται από την επιλογή μιας σουίτας κρυπτογράφησης.

Σουίτα κρυπτογράφησης είναι ένας συνδυασμός αλγορίθμων ελέγχου ταυτότητας, κρυπτογράφησης και κωδικού ελέγχου ταυτότητας μηνυμάτων (MAC) που χρησιμοποιούνται κατά τη χειραψία TLS για τη διαπραγμάτευση των ρυθμίσεων ασφαλείας της σύνδεση. Κάθε σουίτα κρυπτογράφησης ορίζει έναν αλγόριθμο ανταλλαγής κλειδιών (RSA, DH, EDH) όπου καθορίζει τον τρόπο ελέγχου ταυτότητας client/server κατά το handshake, έναν μαζικής κρυπτογράφησης (3DES, AES, RC4) συμπεριλαμβανομένων block ciphers και stream ciphers για την κρυπτογράφηση της ροής μηνυμάτων και έναν κωδικού ελέγχου ταυτότητας μηνυμάτων (SHA-256, SHA-512) που χρησιμοποιείται για τη δημιουργία της σύνοψης μηνύματος.



Εικόνα 3.7: *Παράδειγμα σουίτας κρυπτογράφησης στο TLS 1.3*

Παρακάτω βλέπουμε τις σουίτες κρυπτογράφησης του TLS 1.2 που δεν έχουν καταργηθεί έως τώρα, τον αριθμό εκχώρησης αυτών από το IANA (Internet Assigned Numbers Authority), την αναφορά και το έτος που θα χρησιμοποιούνται ακόμα:

Cipher-Suite	IANA-Nr.	Referenced in	Use until
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	0x00,0x67	[RFC5246]	2028+
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	0x00,0x6B	[RFC5246]	2028+
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	0x00,0x9E	[RFC5288]	2028+
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	0x00,0x9F	[RFC5288]	2028+
TLS_DHE_RSA_WITH_AES_128_CCM	0xC0,0x9E	[RFC6655]	2028+
TLS_DHE_RSA_WITH_AES_256_CCM	0xC0,0x9F	[RFC6655]	2028+

Πίνακας 3.8: *Σουίτες κρυπτογράφησης στο TLS 1.2*

Επίσης στον παρακάτω πίνακα ακολουθούν οι σουίτες κρυπτογράφησης του πρωτοκόλλου TLS 1.3:

Cipher-Suite	IANA-Nr.	Referenced in	Use until
TLS_AES_128_GCM_SHA256	0x13,0x01	[RFC8446]	2028+
TLS_AES_256_GCM_SHA384	0x13,0x02	[RFC8446]	2028+
TLS_AES_128_CCM_SHA256	0x13,0x04	[RFC8446]	2028+

Πίνακας 3.9: *Σουίτες κρυπτογράφησης στο TLS 1.2*

Κεφάλαιο 4
Αλγόριθμοι χαμηλών πόρων

Το NIST (National Institute of Standards Technology) το Μάρτιο του 2019 ξεκίνησε τη διαδικασία για την εύρεση, αξιολόγηση και εν τέλει τυποποίηση ενός ή και περισσοτέρων πιστοποιημένων συστημάτων κρυπτογράφησης με συσχετιζόμενα δεδομένα (AEAD) κατακερματισμού, τα οποία να είναι κατάλληλα για χρήση σε περιβάλλοντα περιορισμένων πόρων (πχ. δίκτυα αισθητήρων), λαμβάνοντάς δε υπόψιν ότι η απόδοση των τρέχων προτύπων κρυπτογράφησης του NIST δεν είναι η κατάλληλη ή είναι ανεπαρκής. Πραγματοποίησε έναν διαγωνισμό με την πρώτη φάση του να έχει συμμετοχή 56 κρυπτογραφικών αλγορίθμων.

Στην συνέχεια, φιναλίστ αποτέλεσαν δέκα αλγόριθμοι οι οποίοι συγκέντρωναν καλύτερα χαρακτηριστικά σε τομείς όπως η ασφάλεια και η αποδοτικότητα σε υλικό και λογισμικό. Αυτοί οι αλγόριθμοι είναι οι εξής: Ο ASCON, ο ELEPHANT, ο GIFT-COFB, ο Grain-128AEAD, ο ISAP, ο PHOTON-Beetle, ο Romulus, ο SPARKLE, ο TinyJAMBU και ο Xoodyak. Η απόφαση για τον αλγόριθμο πρότυπο αποτέλεσε πρόκληση, δεδομένου ότι οι περισσότεροι από τους φιναλίστ παρουσίαζαν σημαντικά πλεονεκτήματα στον τομέα των επιδόσεων, σε σχέση με τα πρότυπα του NIST. Ως νικητήριος αλγόριθμος προς προτυποποίηση αποφασίστηκε ο ASCON καθώς διαπιστώθηκε ότι κάλυπτε τις ανάγκες των περισσότερων χρήσεων όπου απαιτούνταν ελαφριά κρυπτογράφηση.

Οι αλγόριθμοι που έφτασαν στο τελικό στάδιο με τον νικητήριο να μπαίνει πρώτο αναλύονται εκτενώς παρακάτω.

4.1 Αλγόριθμος ASCON

Ο αλγόριθμος ASCON αποτελεί ένα σύστημα αντιμετάθεσης AEAD και hashing. Είναι μια κρυπτογράφηση block cipher με δύο παραλλαγές: Ascon-128 και Ascon-128a. Η αρχική τιμή της κατάστασης αποτελείται από το κλειδί και το nonce. Ο Ascon-128 και ο Ascon-128a έχουν κλειδί 128-bit, nonce 128-bit και ετικέτα 128-bit. Τα μεγέθη μπλοκ δεδομένων του Ascon-128 και του Ascon-128a είναι 64 και 128 bit αντίστοιχα. Όλες αυτές οι παράμετροι είναι σταθερές. Ο ASCON-AEAD βασίζεται στην κατασκευή monkeyDuplex, έναν αμφίδρομο τρόπο λειτουργίας, με κάποιες επιπρόσθετες προσθήκες κλειδιών κατά τις λειτουργίες της αρχικοποίησης και οριστικοποίησης. Οι συναρτήσεις κατακερματισμού του Ascon έχουν ασφάλεια έναντι επιθέσεων σύγκρουσης και επιθέσεων προ-εικόνας και παρέχουν επίσης αντίσταση έναντι επιθέσεων επέκτασης μήκους και επιθέσεων δεύτερης προ-εικόνας. Επιπλέον, παρέχει ισχυρή ασφάλεια έναντι επιθέσεων χρονισμού λόγω των bit-sliced S-boxes του και έχει καλύτερες επιδόσεις, καθώς οι λειτουργίες του βασίζονται σε λέξεις 64-bit και

χρησιμοποιούν μόνο bit-wise πράξεις. Έχουν μετρήσει την απόδοση ως 4,9 με 7,3 Gbps σε υλικό λιγότερο από 10 k GE. Επιπλέον, ο Ascon δεν χρειάζεται καμία αντίστροφη πράξη, γεγονός που τον καθιστά επίσης αποδοτικό. Εκτός από τους ισχυρισμούς για την ασφάλεια και τις επιδόσεις, περιέχει επίσης ένα σκεπτικό σχεδιασμού που παρέχει λεπτομέρειες σχετικά με τις αποφάσεις σχεδιασμού. Για παράδειγμα, η δομή του Ascon η οποία βασίζεται στο Sponge παρέχει πολλά πλεονεκτήματα, όπως αποδεδειγμένη ευελιξία, ασφάλεια και απλότητα.

Κάποιες γνωστές αναλύσεις που παρατίθενται είναι η επίθεση μηδενικού αθροίσματος, η ολοκληρωτική, η γραμμική επίθεση, η διαφορική, η επίθεση μηδενικής συσχέτισης, η αδύνατη διαφορική επίθεση και η επίθεση υποδιαστημικής διαδρομής. Όταν ο αριθμός των γύρων είναι 12, η πολυπλοκότητα αυτών των επιθέσεων είναι πολύ υψηλότερη από 2^128, η οποία είναι αποδεκτή σε περιπτώσεις περιορισμένων πόρων. Εκτός από τις επιθέσεις, αναλύονται επίσης διαφορικές, γραμμικές και αλγεβρικές ιδιότητες. Παρ' όλα αυτά, υπάρχει ένα σαφές κενό σε πρακτικές επιθέσεις πλευρικού καναλιού που δοκιμάστηκαν στον Ascon. Ως εκ τούτου, είναι μια κατεύθυνση για νέα έρευνα σχετικά με τις επιθέσεις πλευρικού καναλιού στον Ascon. Τέλος, ο Ascon περιέχει ένα τμήμα υλοποίησης που περιέχει μετρήσεις σχετικές με την απόδοση, όπως κύκλους ανά byte. Ωστόσο, δεν περιλαμβάνει χαρακτηριστικά επιδόσεων σε συσκευές χαμηλού επιπέδου, όπως μικροελεγκτές των 8-bit. Όταν ο αριθμός των γύρων είναι 12, η πολυπλοκότητα αυτών των επιθέσεων είναι πολύ υψηλότερη από 2^128. Οι αλγόριθμοι κρυπτογράφησης και αποκρυπτογράφησης έχουν μια φάση οριστικοποίησης για την παραγωγή του κώδικα αυθεντικοποίησης μηνύματος (MAC). Η ομάδα Ascon έχει αναπτύξει μια νέα παραλλαγή που ονομάζεται Ascon-80pq, η οποία είναι ασφαλής έναντι της κβαντικής αναζήτησης κλειδιών. Η ειδική παραλλαγή, Ascon-80pq, έχει αντίσταση στις επιθέσεις αναζήτησης κλειδιού με βάση τον αλγόριθμο του Grover, λόγω του αυξημένου μεγέθους κλειδιού. Η υποβολή αποτελείται επίσης από δύο συναρτήσεις κατακερματισμού, δηλαδή τις Ascon-HASH και Ascon-HASHA. Το σύστημα ASCON-Hash στηρίζεται στην κατασκευή duplex sponge construction. Το κύριο συστατικό της οικογένειας ASCON είναι μια αντιμετάθεση 320 bit που ενσαρκώνεται με ανόμοιες σταθερές και αριθμό γύρων για διαφορετικές παραλλαγές. Έχει σχεδιαστεί για να αποδίδει καλά σε υλικό αλλά και σε λογισμικό. Ο αλγόριθμος επιλέχθηκε ως η πρώτη επιλογή για την ελαφριά κρυπτογράφηση με την λειτουργία της αυθεντικοποίησης στο τελικό χαρτοφυλάκιο του διαγωνισμού CAESAR. (2014-2019).

Το 2013, το CAESAR (ο ``Διαγωνισμός για την Αυθεντική Κρυπτογράφηση: Security, Applicability, and Robustness'') που ιδρύθηκε από το NIST και τον Dan Bern stein με στόχο την αναζήτηση και εύρεση συστημάτων αυθεντικής κρυπτογράφησης, τα οποία να προσφέρουν πλεονεκτήματα έναντι του AES-GCM και να είναι κατάλληλα για ευρεία υιοθέτηση. Ο πρώτος γύρος ξεκίνησε τον Μάρτιο του 2014 με συμμετοχή με 57 υποψηφίων εκ των οποίων εννέα από αυτούς δεν μπόρεσαν να ανταποκριθούν και αποσύρθηκαν από τον διαγωνισμό. Οι υπόλοιποι 48 υποψήφιοι πέρασαν από μια εντατική διαδικασία εξέτασης,

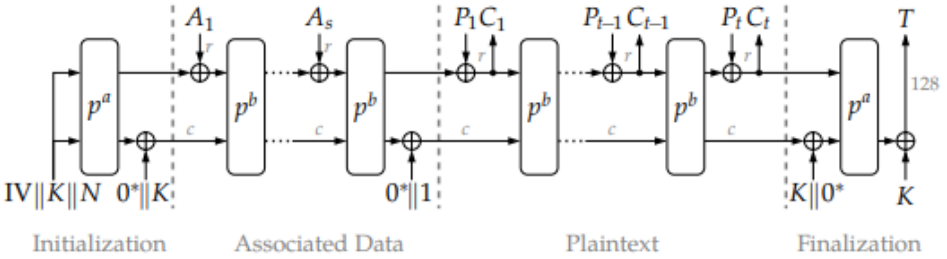
ανάλυσης και σύγκρισης. Μετά από αυτήν την εντατική εξέταση και ανάλυση των 48 υποψηφίων από την κοινότητα, η επιτροπή CAESAR επέλεξε μόνο 30 υποψηφίους για τον δεύτερο γύρο. Η ανακοίνωση για τους υποψηφίους του τρίτου γύρου έγινε στις 15 Αυγούστου 2016 και 15 υποψήφιοι επιλέχθηκαν για τον τρίτο γύρο. Υποβλήθηκαν τέσσερα κρυπτογραφήματα στο CAESAR: ACORN, AEGIS, MORUS και JAMBU. Τα AEGIS και MORUS λόγω του ότι ήταν γρήγορα σε λογισμικό και τα ACORN και JAMBU διότι ήταν για ελαφρές εφαρμογές. Τα ACORN, AEGIS, MORUS απαιτούσαν μοναδικό nonce για κάθε μήνυμα και το JAMBU παραμένει λογικά ισχυρό όταν το nonce χρησιμοποιείται καταχρηστικά. Τα κρυπτογραφήματα ACORN, AEGIS επιλέχθηκαν εν τέλει ως νικητές του διαγωνισμού CAESAR τον Μάρτιο του 2019. Υπήρξαν έξι νικητές. Για την περίπτωση χρήσης σε ελαφρές εφαρμογές αναδείχθηκε ο ASCON και ο ACORN. Για την περίπτωση χρήσης σε εφαρμογές υψηλής απόδοσης επιλέχθηκε ο AEGIS-128 και ο OCB. Για την περίπτωση χρήσης με άμυνα σε βάθος επιλέχθηκαν ο DEOXOYS-II και ο COLM.

Κάποια από τα χαρακτηριστικά του εν λόγω αλγορίθμου είναι τα παρακάτω:

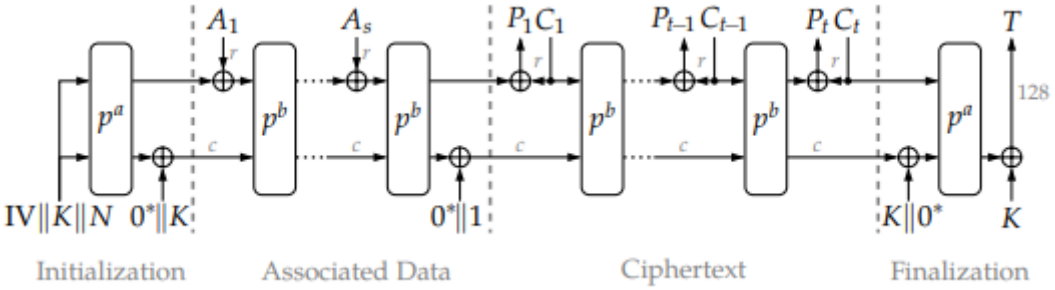
- Επαληθευμένη κρυπτογράφηση και κατακερματισμός (σταθερό ή μεταβλητό μήκος εξόδου) με μία ελαφριά μετάθεση.
- Τρόποι λειτουργίας με βάση το σφουγγάρι με προσαρμοσμένη μετάθεση SPN
- Αποδεδειγμένα ασφαλής λειτουργία με κλείδωμα οριστικοποίησης για πρόσθετη στιβαρότητα
- Εύκολη εφαρμογή σε λογισμικό και υλικό
- Ελαφρύς για συσκευές περιορισμένων πόρων: μικρή κατάσταση, απλή μετάθεση, ισχυρή λειτουργία
- Γρήγορος σε υλικό
- Γρήγορος σε λογισμικό: αγωγιμοποιήσιμος, 5-bit S-box για αρχιτεκτονικές 64 bit
- Επεκτάσιμος για μεγαλύτερη ασφάλεια ή υψηλότερη απόδοση
- Αντίσταση χρονισμού
- Αντίσταση πλευρικού καναλιού: S-box βελτιστοποιημένο για αντίμετρα
- Μέγεθος κλειδιού = μέγεθος ετικέτας = επίπεδο ασφάλειας (συνιστάται 128 bit)
- Ελάχιστη επιβάρυνση (μήκος κρυπτογραφημένου κειμένου = μήκος απλού κειμένου)
- Single pass, online (κρυπτογράφηση και αποκρυπτογράφηση), nonce-based, inverse-free

AEAD variants	*Key*	*Nonce*	*Tag*	*# Rounds*	Hash variants	*Digest size*	*# Rounds*
ASCON-128	128	128	128	12, 6, 12	**ASCON-Hash**	256	12
ASCON-128a	128	128	128	12, 8, 12	ASCON-Xof	256	12
ASCON-80pq	160	128	128	12, 6, 12			

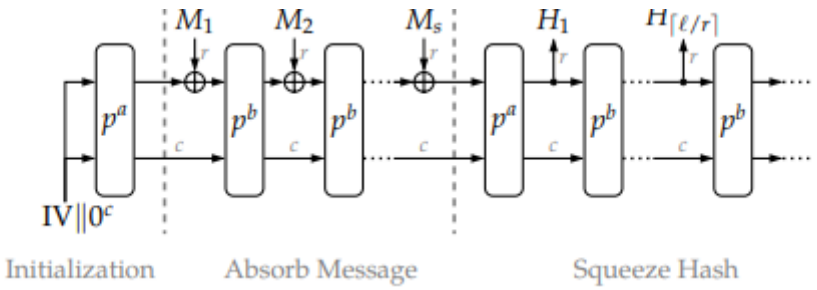
Πίνακας 4.1: *Χαρακτηριστικά του ASCON*



Σχήμα 4.2 : Κρυπτογράφηση του ASCON



Σχήμα 4.3: Αποκρυπτογράφηση του ASCON



Σχήμα 4.4: Κατακερματισμός του ASCON

4.2 Αλγόριθμος ELEPHANT

Ο αλγόριθμος Elephant είναι ο δεύτερος στη λίστα των φιναλίστ του LWC. Αποτελεί έναν block cipher lightweight αλγόριθμο κρυπτογράφησης με έλεγχο ταυτότητας. Η λειτουργία του βασίζεται σε μετάθεση και inverse-free. Ωστόσο, διαθέτει μικρό μέγεθος κατάστασης και επιτρέπει υψηλό βαθμό παραλληλισμού. Κάνει χρήση LFSR για κάλυψη (masking) και είναι αποτελεσματικός με τον τρόπο που πρέπει να γίνει ακριβώς η κάλυψη.

Η λειτουργία του Elephant είναι μια encrypt-then-MAC με βάση το nonce, όπου η κρυπτογράφηση εκτελείται με λειτουργία μετρητή (counter mode) και ο έλεγχος ταυτότητας μηνύματος από μια παραλλαγή του προστατευμένου αθροίσματος μετρητή. Επίσης, η προαναφερθείσα λειτουργία χρησιμοποιείται για την αυθεντικοποίηση. Και τα δύο, έμμεσα δημιουργούνται, χρησιμοποιώντας μια απλοποίηση του μπλοκ κρυπτογράφησης με δυνατότητα προσαρμογής με μάσκα Even-Mansour του Granger. Ο Elephant είναι ο μόνος που χρησιμοποιεί παράλληλη λειτουργία με μετάθεση και ο τρόπος λειτουργίας γίνεται χρησιμοποιώντας τις μεταθέσεις του Spongent και του KECCAK. Ο σχεδιασμός του Elephant χωρίς αντίστροφο σχεδιασμό είναι παραλληλοποιήσιμος σε λογισμικό και υλικό. Διαθέτει τρεις παραλλαγές, Dumbo, Jumbo και Delirium. Οι κύριες παραλλαγές, Dumbo και Jumbo, βασίζονται στον κατακερματισμό Spongent, ενώ το Delirium χρησιμοποιεί το Keccak ως πρωταρχικό στοιχείο. Και οι τρεις παραλλαγές χρησιμοποιούν LFSR για τη συγκάλυψη. Επιπλέον, οι Dumbo, Jumbo και Delirium έχουν μεγέθη μπλοκ 160 bit, 176 bit και 200 bit αντίστοιχα. Το κύριο masking που χρησιμοποιείται στην αυθεντική κρυπτογράφηση αποτελείται από ένα LFSR και μια αντιμετάθεση (Spongent ή Keccak). Αυτό θα γίνει με XOR με το απλό κείμενο για να παραχθεί το κρυπτογραφημένο κείμενο στην κρυπτογράφηση, και το κρυπτογραφημένο κείμενο θα γίνει XOR με τη μάσκα για να ανακτηθεί το απλό κείμενο. Ο Elephant παρουσιάζει προδιαγραφές για τις μεταθέσεις και για τις τρεις παραλλαγές - το μέγεθος της ετικέτας για τα Dumbo και Jumbo είναι 64 bit, ενώ για το Delirium είναι 128 bit. Επίσης ο Elephant περιέχει λεπτομερείς αναλύσεις της τυπικής ασφάλειας πολλαπλών χρηστών πιστοποιημένου τρόπου κρυπτογράφησης. Οι εκδόσεις Dumbo, Jumbo και Delirium επιτυγχάνουν ασφάλεια 112-bit, 127-bit και 127-bit αντίστοιχα. Το κομμάτι ανάλυσης ασφάλειας του Elephant περιέχει μια θεωρητική ανάλυση της διαφορικής, γραμμικής και ολοκληρωτικής κρυπτανάλυσης στις μεταθέσεις Spongent και Keccak.

Ωστόσο, χρειάζεται περισσότερες πληροφορίες σχετικά με την κρυπτανάλυση τρίτου μέρους στον κρυπτογράφο. Δεν υπάρχουν λεπτομέρειες σχετικά με τις επιθέσεις πλευρικού καναλιού στον Elephant. Ωστόσο, χρειάζεται περισσότερη ανάλυση ασφαλείας από τρίτους.

Ο Elephant αποτελείται από τρεις περιπτώσεις:

1. **Dumbo: Elephant-Spongent-π.** Επιτυγχάνει ασφάλεια 112 bit με τη προϋπόθεση ότι η διαδικτυακή πολυπλοκότητα (online complexity) είναι

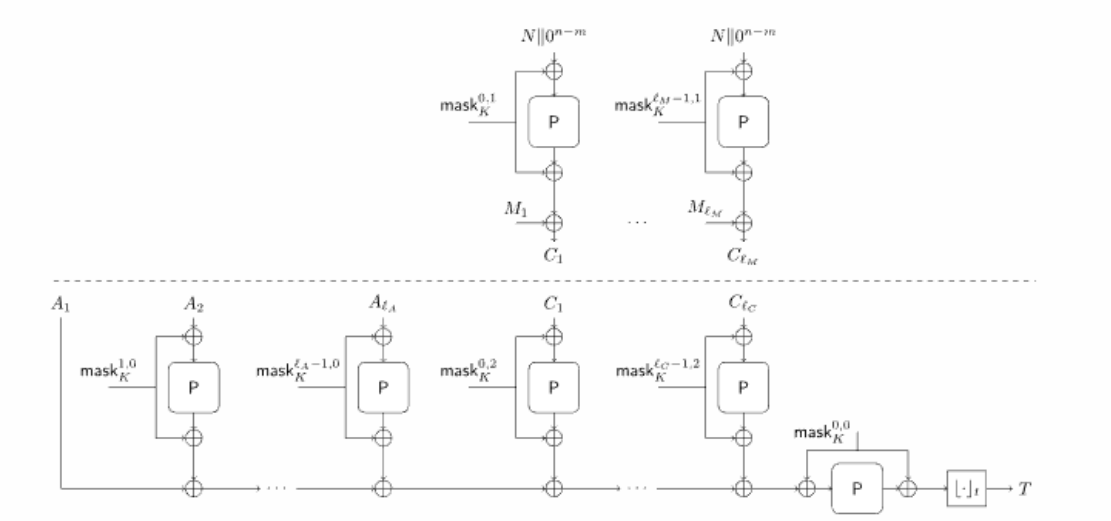
το πολύ περίπου 246 μπλοκ. Η περίπτωση αυτή είναι κατάλληλη για υλικό, όπως είναι το ίδιο το Spongent.

2. **Jumbo: Elephant-Spongent-π.** Επιτυγχάνει ασφάλεια 127 bit υπό τις ίδιες συνθήκες στην διαδικτυακή πολυπλοκότητα. Είναι μια ελαφρώς πιο συντηρητική περίπτωση του Elephant, όπου βασίζεται στην ίδια οικογένεια μεταθέσεων.
3. **Delirium: Elephant-Keccak-f**. Επιτυγχάνει ασφάλεια 127-bit, με υψηλότερο όριο περίπου 270 μπλοκ online complexity και δημιουργήθηκε με Keccak-f. Αυτή η παραλλαγή έχει αναπτυχθεί περισσότερο για χρήση λογισμικού αλλά αποδίδει αρκετά καλά σε υλικό. Η μετάθεση είναι η μικρότερη περίπτωση που καθορίζεται στο πρότυπο NIST SHA-3.

Παρά την υποστήριξή παραλληλισμού, όλες οι εκδόσεις του Elephant έχουν μικρό μέγεθος κατάστασης (state size).

Algorithm	Member	Block size	Key size	Permutation	Tag	Nonce	Non-linear operation
Elephant	Delirium	200	128	200bit - KECCAK	128	96	Boolean function
	Dumbo	160	128	160bit - Spongent	64	96	s-box 8bit
	Jumbo	176	128	176bit - Spongent	64	96	s-box 8bit

Πίνακας 4.5: Χαρακτηριστικά του αλγορίθμου Elephant



Σχήμα 4.6*:* Τρόπος λειτουργίας Elephant.

4.3 Αλγόριθμος GIFT-COFB

Ο αλγόριθμος GIFT-COFB, είναι ένα σύστημα AEAD με ρυθμό 1 (block cipher GIFT-128) που βασίζεται σε 128-bit. Ο τρόπος λειτουργίας του GIFT-COFB διαφέρει σε μερικά σημεία από την αρχική λειτουργία COmbined FeedBack (COFB). Όταν ένα μπλοκ κρυπτογράφησης 128-bit χρησιμοποιείται, ο αρχικός τρόπος COFB χρησιμοποιεί ένα nonce 64-bit και ένα feedback, το οποίο δεν υφίσταται απώλεια εντροπίας. Εναλλακτικά, ο τρόπος GIFT-COFB χρησιμοποιεί ένα nonce 128-bit, πολύ αποδοτικό από πλευράς υλικού ανατροφοδότησης με απώλεια εντροπίας 1 bit και χειρίζεται κενά δεδομένα αλλάζοντας τη μέθοδο συμπλήρωσης. Ο αλγόριθμος δεν απαιτεί ακριβές αντίστροφες πράξεις για την αποκρυπτογράφηση. Οι συνιστώμενες παράμετροι είναι ένα μπλοκ 128 bit και μια ετικέτα 128 bit. Το κρυπτογραφικό πρωτεύον που χρησιμοποιείται στο GIFT-COFB είναι το GIFT-128. Πρόκειται για ένα κρυπτογράφημα δικτύου αντικατάστασης-παρεμβολής (SPN) 40 γύρων με κλειδί 128-bit. Η κύρια λειτουργία γύρου του GIFT-128 αποτελείται από τέσσερις κύριες φάσεις: αρχικοποίηση, αντικατάσταση κελιών, αντιμετάθεση bits και προσθήκη κλειδιού γύρου. Απαιτεί επίσης έναν μηχανισμό χρονοπρογραμματισμού κλειδιών και σταθερές γύρου. Επιπλέον, το GIFT-128 διαθέτει μια παραλλαγή βασισμένη σε πίνακα αναζήτησης (LUT) για τη βελτίωση του χρόνου εκτέλεσης με τη χρήση περισσότερου χώρου.

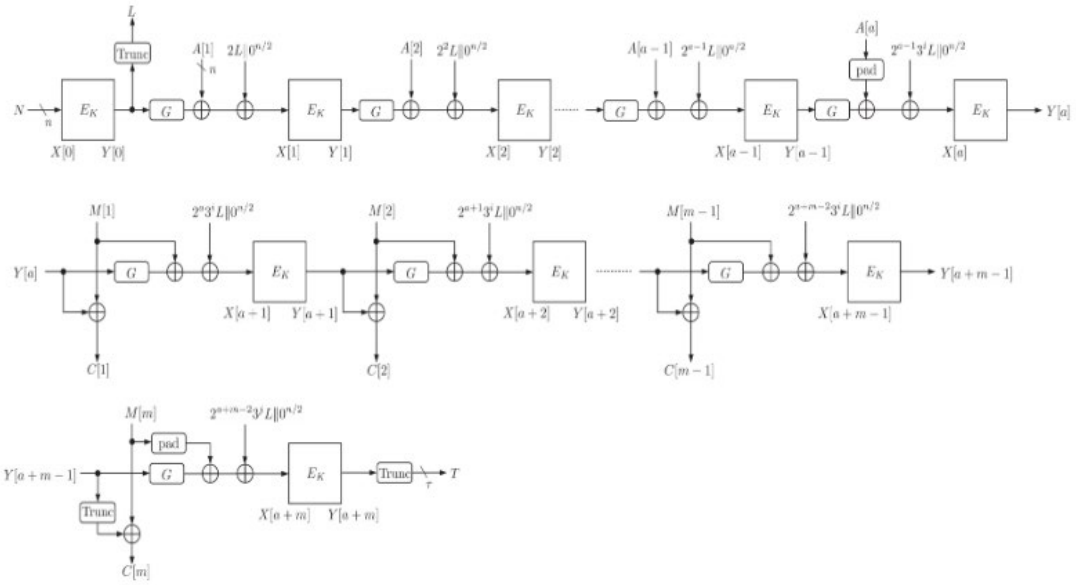
Ο GIFT-COFB λαμβάνει συσχετιζόμενα δεδομένα A αυθαίρετου μήκους, μήνυμα M (απλό κείμενο) αυθαίρετου μήκους ως είσοδο και επιστρέφει κρυπτοκείμενο C ίδιου μήκους με του μηνύματος, και μία ετικέτα (tag) 128 bit.

Χαρακτηριστικά:

- **Βασίζεται σε λειτουργία επικυρωμένης κρυπτογράφησης (Authenticated Encryption) με υψηλό ρυθμό (rate) αλλά μικρή μνήμη**: Μπορεί να υλοποιηθεί με πολύ χαμηλό μέγεθος κατάστασης (state size) στα 1.5n+k (n=state size, k=key length) καθώς επιτυγχάνει βέλτιστο ρυθμό 1.
- **Λειτουργία υψηλής ευελιξίας:** μπορεί εύκολα να προσαρμοστεί οποιοδήποτε μπλοκ κρυπτογράφησης στη δομή του, όπου όταν χρησιμοποιείται με ελαφρύτερους block cipher, έχει χαμηλότερη κατανάλωση αποτυπώματος (footprint) υλικού.
- **Χαμηλή επιβάρυνση(Low Overhead):** Εκτός από την κλήση block cipher, απαιτεί μόνο 5n/2-bit XOR ανά μπλοκ δεδομένων + 1-bit δεξιά περιστροφή (rotation), n/4-bit κατάσταση, το οποίο δείχνει να είναι πολύ χαμηλό overhead.
- **Υψηλό όριο ασφαλείας**: Η χρήση συνδυασμένης ανατροφοδότησης (combined feedback) ανεβάζει το επίπεδο ασφάλειας. Το όριο αυξήθηκε σχεδόν σε όριο γενεθλίων.
- **Inverse-Free:** Ο αλγόριθμος κρυπτογράφησης όσο και αποκρυπτογράφησης δεν απαιτούν καμία κλήση αποκρυπτογράφησης

στον υποκείμενο block cipher. Αυτό μειώνει σημαντικά το συνολικό αποτύπωμα υλικού σε συνδυασμένες υλοποιήσεις κρυπτογράφησης-αποκρυπτογράφησης.

- **Χαμηλός αριθμός κλήσεων κρυπτογράφησης μπλοκ:** Απαιτεί μόνο a+m+1 πολλές πρωτόγονες επικλήσεις για την επεξεργασία ενός μπλοκ συσχετιζόμενων δεδομένων (AD) και ενός μπλοκ μηνύματος m.
- **Αποδοτικότητα σύντομων μηνυμάτων**: Η βέλτιστη απόδοση στον αριθμό των κλήσεων και το χαμηλό overhead, βοηθούν να έχει πολύ υψηλή απόδοση για μικρά μηνύματα.



Σχήμα 4.7*:Λειτουργία του GIFTCOFB*

4.4 Αλγόριθμος Grain-128AEAD

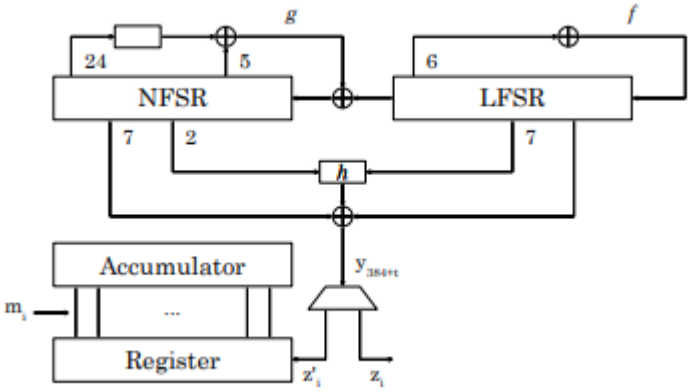
Ο Grain-128AEAD είναι ένας αλγόριθμος stream cipher, μέλος της οικογένειας κρυπτογραφικών αλγορίθμων Grain, και αποτελεί έναν bit-oriented καταχωρητή μετατόπισης, με ανατροφοδότηση, η οποία βασίζεται σε ένα σύστημα AEAD βελτιστοποιημένο, έτοιμο να ανταποκριθεί σε υλοποιήσεις υλικού. Το 2008, το Grainv1 επιλέχθηκε ως το ανώτερο στο προφίλ υλικού του χαρτοφυλακίου eSTREAM. Το πρότυπο ISO/IEC 29167-13:2015 περιλαμβάνει το Grain-128a για συστήματα RFID.

Ο σχεδιασμός του Grain-128AEAD είναι ελαφρώς διαφορετικός από τις προηγούμενες εκδόσεις, ο οποίος έχει ετικέτα (Tag) 64 bit, nonce (IV) σταθερού μήκους-μεγέθους 96 bit και ένα κλειδί σταθερού μήκους-μεγέθους 128 bit. Δύο βασικά στοιχεία από τα οποία και αποτελείται είναι: μια γεννήτρια προ-εξόδου, η οποία κατασκευάζεται από τη χρήση ενός καταχωρητή μετατόπισης γραμμικής ανάδρασης (LFSR), ενός καταχωρητή μετατόπισης μη γραμμικής ανάδρασης (NFSR) και μιας συνάρτησης προ-εξόδου, και ως δεύτερο μια γεννήτρια ελέγχου ταυτότητας που αποτελείται από έναν συσσωρευτή και έναν καταχωρητή

μετατόπισης. Στη φάση αρχικοποίησης, το NFSR και το LFSR αρχικοποιούνται με το κλειδί και το nonce. Μετά την αρχικοποίηση, η γεννήτρια προ-εξόδου παράγει μια ροή bit που αργότερα γίνεται XOR με το απλό κείμενο για να παραχθεί το κρυπτογράφημα. Ένα σημαντικό χαρακτηριστικό που αναφέρεται στο Grain-128AEAD είναι ότι υποστηρίζει μια μάσκα AEAD. Μας επιτρέπει να ορίσουμε ποια bits είναι απλό κείμενο και ποια bits είναι πρόσθετα δεδομένα. Αυτό θα είναι σημαντικό κατά το σχεδιασμό νέων πρωτοκόλλων επικοινωνίας, καθώς η θέση της επικεφαλίδας μπορεί να καθοριστεί εύκολα.

Το Time/Memory/Data trade-off (TMD-TO) είναι ένας τύπος επίθεσης από τον οποίο πολλοί κρυπτογράφοι ροής δεν είναι ασφαλείς. Η εσωτερική κατάσταση του Grain-128AEAD δεν μπορεί να ανακατασκευαστεί- ως εκ τούτου, μια επίθεση TMD-TO έχει πολυπλοκότητα 2128. Είναι επίσης ασφαλής έναντι αλγεβρικών επιθέσεων λόγω του μεγάλου αλγεβρικού βαθμού της συνάρτησης εξόδου. Ένα επαναχρησιμοποιούμενο ζεύγος κλειδιού/μηδενικού μπορεί να προκαλέσει διαρροή πληροφοριών σχετικά με το απλό κείμενο. Ως εκ τούτου, δεν επιτρέπεται στο Grain-128AEAD. Ο Grain-128AEAD διαθέτει αντίμετρα κατά των επιθέσεων συσχέτισης. Ο λόγος είναι ότι διαθέτει μια μεγάλη κατάσταση. Ωστόσο, αναφέρεται ότι το κρυπτογράφημα Grain-128 έχει παραβιαστεί με τη χρήση μιας δυναμικής επίθεσης κύβου. Οι ερευνητές αναμένουν ότι η βελτιωμένη διαδικασία αρχικοποίησης θα αποτρέψει τις επιθέσεις κύβου και ανάκτησης κλειδιού και ισχυρίζονται ότι η επίθεση ανάκτησης κλειδιού έχει πολυπλοκότητα 2^96. Έχουν πραγματοποιηθεί μερικές επιτυχημένες επιθέσεις σφάλματος κατά του Grain.

Ο σχεδιασμός του Grain-128AEAD με ένα μόνο bit-by-bit είναι ιδιαίτερα αποδοτικός σε υλοποιήσεις υλικού. Οι ερευνητές έχουν υλοποιήσει τον αλγόριθμο χρησιμοποιώντας το stm065v536. Για τη σύνθεση και την προσομοίωση χρησιμοποιείται το Synopsys Design Compiler 2013. Έχουν καταγράψει τον απαιτούμενο αριθμό πυλών ως 3638,5 GE. Όταν το επίπεδο παραλληλισμού είναι 32, ο αριθμός των πυλών είναι 12.110,5. Επιπλέον, η απαιτούμενη ισχύς του υλικού είναι 313 nW και ο ρυθμός μετάδοσης είναι 50 kbit/s. Η πρακτική κρυπτανάλυση των επιθέσεων πλευρικού καναλιού είναι μια ανοικτή κατεύθυνση για την έρευνα κατά του Grain-128AEAD.



Σχήμα 4.8: Λειτουργία του Grain 128AEAD.

4.5 Αλγόριθμος ISAP

Ο αλγόριθμος ISAP, είναι ένα σύστημα AEAD βασισμένο στην αντιμετάθεση, η οποία σχεδιάστηκε για να παρέχει, από το αλγοριθμικό επίπεδο, ασφάλεια έναντι ενός ευρύτερου φάσματος επιθέσεων εφαρμογής, όπως επιθέσεις διαφορικού σφάλματος, επιθέσεις στατιστικού σφάλματος, στατιστικού αναποτελεσματικού σφάλματος και διαφορικής ανάλυσης ισχύος. Όσον αφορά τον τρόπο λειτουργίας του ISAP, είναι μια nonce-based encrypt-then-MAC κατασκευή, όπου η κρυπτογράφηση πραγματοποιείται με XOR ενός μηνύματος και ενός keystream και η αυθεντικοποίηση/επαλήθευση βασίζεται σε ένα παράδειγμα hash-then-MAC.

Η οικογένεια ISAP έχει τέσσερις παραλλαγές, οι οποίες είναι οι ISAP-A-128A, ISAP-K-128A, ISAP-A-128 και ISAP-K-128. Οι ISAP-A-128A και ISAP-A-128 χρησιμοποιούν το Ascon-p ως πρωτότυπο, ενώ οι ISAP-K-128A και ISAP-K-128 χρησιμοποιούν το Keccak-p. Όλα τα μέλη του ISAP έχουν μέγεθος κλειδιού 128 bit. Ο ISAP-A-128-A και ο ISAP-A-128 αποτελούνται από μια κατάσταση 320-bit, ενώ ο ISAP-K-128A και ο ISAP-K-128 έχουν καταστάσεις 400-bit.

Ο ρυθμός αναπαρίσταται ως πλειάδα a, και b, όπου το a αντιπροσωπεύει το μέγεθος του ρυθμού για την επεξεργασία nonce στη συνάρτηση επανακλειδώματος IsapRk και το b αντιπροσωπεύει το μέγεθος του ρυθμού για όλες τις άλλες φάσεις. Ο αριθμός των γύρων αναπαρίσταται με τη χρήση μιας τετραπλής sH , sB, sE , sK που δείχνει τον αριθμό των γύρων της μετάλλαξης που χρησιμοποιείται κατά τη φάση της πιστοποίησης ταυτότητας, τη φάση της επεξεργασίας nonce, τις φάσεις κρυπτογράφησης και αποκρυπτογράφησης και τη δημιουργία κλειδιών συνόδου στη φάση της συνάρτησης επανακλειδώματος, αντίστοιχα.

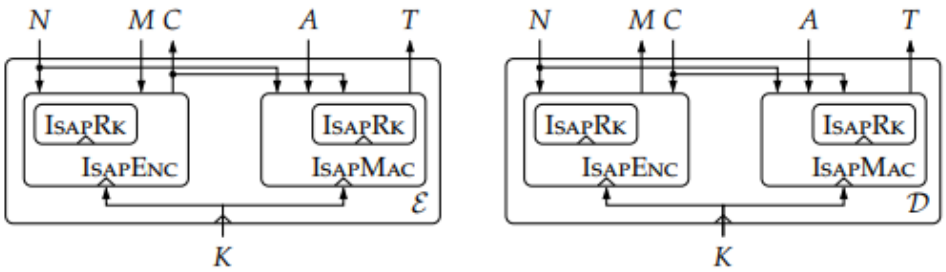
Οι παραλλαγές της οικογένειας ISAP παρατίθενται παρακάτω.

AEAD variants	*Key*	*Nonce*	*Tag*	*Permutation*	*Rate*	*#Rounds*
ISAP-K-128a	128	128	128	400-bit KECCAK	144,1	16,1,8,8
ISAP-A-128a	128	128	128	320-bit ASCON	64,1	12,1,6,12
ISAP-K-128	128	128	128	400-bit KECCAK	144,1	20,12,12,12
ISAP-A-128	128	128	128	320-bit ASCON	64,1	12,12,12,12

Πίνακας 4.9: Παραλλαγές -Χαρακτηριστικά του ISAP.

Κάποια από τα χαρακτηριστικά του συγκεκριμένου αλγορίθμου είναι τα εξής:

- Επικυρωμένη κρυπτογράφηση με χρήση ελαφρών μεταθέσεων
- Λειτουργία που βασίζεται σε σφουγγάρι χρησιμοποιώντας μεταθέσεις SPN
- Κατάλληλο για συσκευές περιορισμένων πόρων: μικρή κατάσταση, απλή αντιμετάθεση
- Αντίσταση πλευρικού καναλιού: Αποδεδειγμένα ασφαλής ανθεκτικότητα διαρροής για κρυπτογράφηση/αποκρυπτογράφηση
- Ενσωματωμένη σκλήρυνση έναντι επιθέσεων σφαλμάτων
- Εύκολη εφαρμογή σε λογισμικό και υλικό
- Ανθεκτικός σε λογισμικό: αγωγιμοποιήσιμο, χωρισμένο σε κομμάτια S-box 5 bit
- Γρήγορος και ανθεκτικό σε υλικό
- Επεκτάσιμος για μεγαλύτερη ασφάλεια ή υψηλότερη απόδοση
- Αντίσταση χρονισμού
- Ελάχιστη επιβάρυνση (μήκος κρυπτογραφημένου κειμένου = μήκος απλού κειμένου)



Σχήμα 4.10: Κρυπτογράφηση/Αποκρυπτογράφηση του ISAP.

4.6 Αλγόριθμος Photon-Beetle

Photon-Beetle είναι μία οικογένεια αλγορίθμων επικυρωμένης κρυπτογράφησης (AD) και κατακερματισμού, όπου χρησιμοποιεί λειτουργία Beetle που βασίζεται σε σφουγγάρι (Sponge) και P256 ως υποκείμενη μετάθεση (PHOTON256).

Ο αλγόριθμος ταξινομείται σε δύο κατηγορίες: α) αυθεντικών κρυπτογραφήσεων PHOTON-Beetle-AEAD που βασίζεται σε AEAD που μοιάζει με σφουγγάρι σε συνδυασμό με feedback και β) συναρτήσεων κατακερματισμού PHOTON-Beetle-Hash που βασίζεται σε μία δομή σφουγγαριού.

Ο PHOTON-Beetle-AEAD λαμβάνει ως είσοδο α) ένα κλειδί κρυπτογράφησης 128-bit K, β)ένα nonce 128-bit N, γ) συσχετιζόμενα δεδομένα A αυθαίρετου μήκους, γ) ένα μήνυμα M αυθαίρετου μήκους και επιστρέφει ένα κρυπτογραφημένο κείμενο C ίδιου μήκους με αυτό του μηνύματος και μια ετικέτα (tag) 128-bit T. Ο PHOTON-Beetle-Hash[r] λαμβάνει ένα μήνυμα M αυθαίρετου μήκους και επιστρέφει μια ετικέτα 256-bit T.

AEAD variants	*Key*	*Nonce*	*Tag*	*Rate*	Hash variants	*Digest size*	*Rate*
PHOTON-Beetle-AEAD[128]	128	128	128	128/128	**PHOTON-Beetle-Hash[32]**	256	32/128
PHOTON-Beetle-AEAD[32]	128	128	128	32/128			

Πίνακας 4.11: Παραλλαγές του Photon-Beetle.

Παρακάτω ακολουθούν χαρακτηριστικά του αλγόριθμου Photon-Beetle:

- **Υψηλό όριο ασφαλείας:** καθιστά τη λειτουργία ελαφριά, ελαχιστοποιώντας το μέγεθος της κατάστασης (state). Στην πραγματικότητα, γίνεται χρήση μόνο μιας αντιμετάθεση 256-bit.
- **Υψηλά ευέλικτη λειτουργία:** Μπορεί εύκολα να εφαρμόσει οποιαδήποτε μετάθεση σε αυτή τη δομή. Αυτό δείχνει ότι, όταν γίνεται χρήση με ελαφρύτερες μεταθέσεις, καταναλώνει μικρότερο αποτύπωμα υλικού (hardware footprint).
- **Ενιαία κατάσταση**: Το μέγεθος κατάστασης (state size)είναι αρκετά μικρό όσο το μέγεθος του μπλοκ της υποκείμενης μετάθεσης όπου αυτό εξασφαλίζει καλά χαρακτηριστικά υλοποίησης τόσο σε ελαφριές όσο και σε πλατφόρμες υψηλής απόδοσης.
- **Inverse-Free**: ο αλγόριθμος κρυπτογράφησης όσο και αποκρυπτογράφησης δεν απαιτούν κλήση αποκρυπτογράφησης στον υποκείμενο tweakable block cipher. Αυτό μειώνει σε μεγάλο βαθμό το συνολικό αποτύπωμα υλικού σε υλοποιήσεις κρυπτογράφησης-αποκρυπτογράφησης**.**
- **Σχεδόν χωρίς επιβάρυνση:** Εκτός από τη μετάθεση, απαιτεί μόνο 2r-bit XOR ανά μπλοκ δεδομένων + 1-bit δεξιά περιστροφή (rotation) μιας κατάστασης r/2-bit, όπου φαίνεται η επιβάρυνση να είναι πολύ μικρή.
- **Βέλτιστος**: Ο PHOTON-Beetle-Hash απαιτεί μόνο (m-3) πολλές primitive calls για την επεξεργασία ενός μηνύματος m μπλοκ. Δεσμεύει 4 μπλοκ r-bit μαζί στον πρώτο κύκλο ρολογιού. Ο AEAD απαιτεί μόνο a+m+1 κλήσεις αντιμετάθεσης για ένα a μπλοκ συνδεδεμένων δεδομένων και ένα m μπλοκ μηνύματος.
- **Αποδοτικότητα σε μικρά μηνύματα**: η συνάρτηση κατακερματισμού απορροφά τα πρώτα 128 bit απλού κειμένου (plaintext) ως αρχικό διάνυσμα (iv), το οποίο έχει στόχο να είναι αποτελεσματικό για μικρό μήνυμα.

4.7 Αλγόριθμος Romulus

Ο αλγόριθμος Romulus, είναι ένα σύστημα AEAD που βασίζεται στο tweakable block κρυπτογράφησης SKINNY. Αποτελείται από δύο οικογένειες: α)βασισμένος σε nonce AEAD RomulusN και β) ανθεκτικός σε κατάχρηση nonce AEAD Romulus-M. Ο Romulus-N χρησιμοποιεί ένα TBC βασισμένο σε ρυθμό 1 (rate) και ο τρόπος λειτουργίας του Romulus-M ακολουθεί μια προσέγγιση MAC-then-encrypt.

Ο Romulus αποτελείται από 4 παραλλαγές, καθεμία από τις οποίες χρησιμοποιεί εσωτερικά το προσαρμόσιμο μπλοκ κρυπτογράφησης Skinny-128/384+. Οι παραλλαγές είναι οι εξής:

1. Romulus-N, ένα AEAD (NAE) το οποίο δεν βασίζεται σε καμία περίπτωση
2. Romulus-M, AEAD (MRAE) το οποίο δεν αντέχει στη χρήση
3. Romulus-T, ένα AEAD το οποίο είναι ανθεκτικό σε διαρροές
4. Romulus-H, η οποία είναι μία συνάρτηση κατακερματισμού

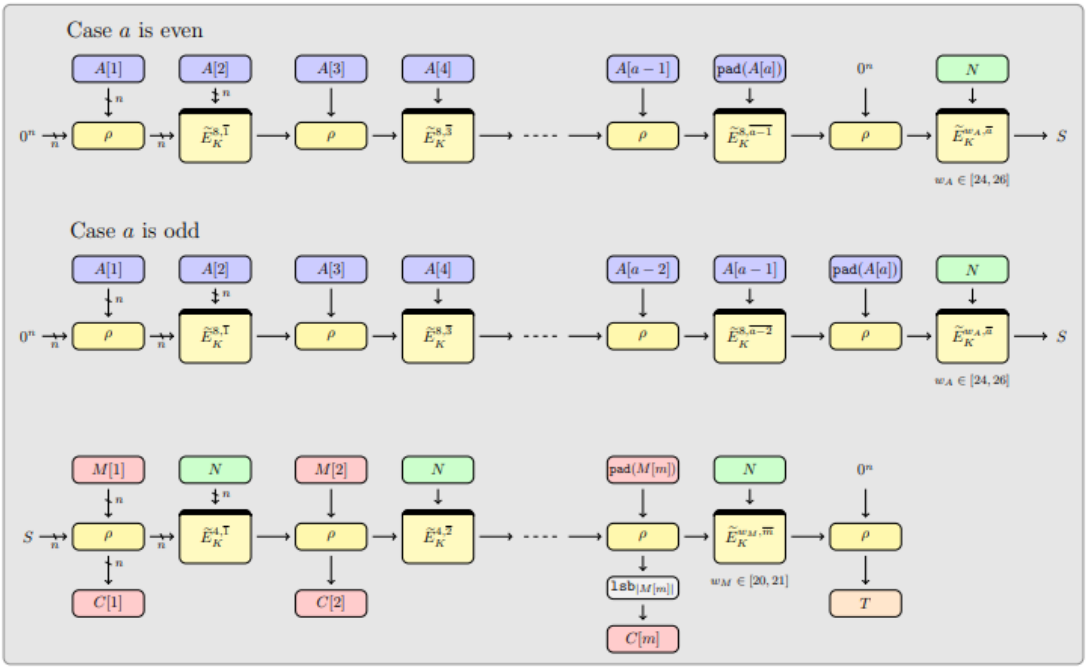
Μεταξύ αυτών, το Romulus-N είναι η κύρια παραλλαγή. Ο τρόπος λειτουργίας της δομής Romulus είναι πολύ παρόμοιος με τον COFB. Και οι τρεις παραλλαγές της κρυπτογράφησης Romulus έχουν ένα κλειδί 128-bit, ένα nonce 128-bit και μια ετικέτα 128-bit. Το μέγεθος του μπλοκ δεδομένων είναι 128 bit. Επιτρέπει την κρυπτογράφηση/αποκρυπτογράφηση 2^59 bytes δεδομένων, συμπεριλαμβανομένων των σχετικών δεδομένων, χρησιμοποιώντας ένα μόνο ζεύγος key-value. Επιπλέον, ο Romulus-H παράγει τιμές κατακερματισμού 256-bit για οποιαδήποτε είσοδο.

Όπως ισχυρίζονται οι σχεδιαστές, τα Romulus-N και Romulus-M παρέχουν ασφάλεια 128-bit. Το Romulus-M παρέχει ασφάλεια 128-bit ακόμη και αν το nonce επαναχρησιμοποιηθεί. Επιπλέον, το Romulus-T προσφέρει ασφάλεια 121-bit.

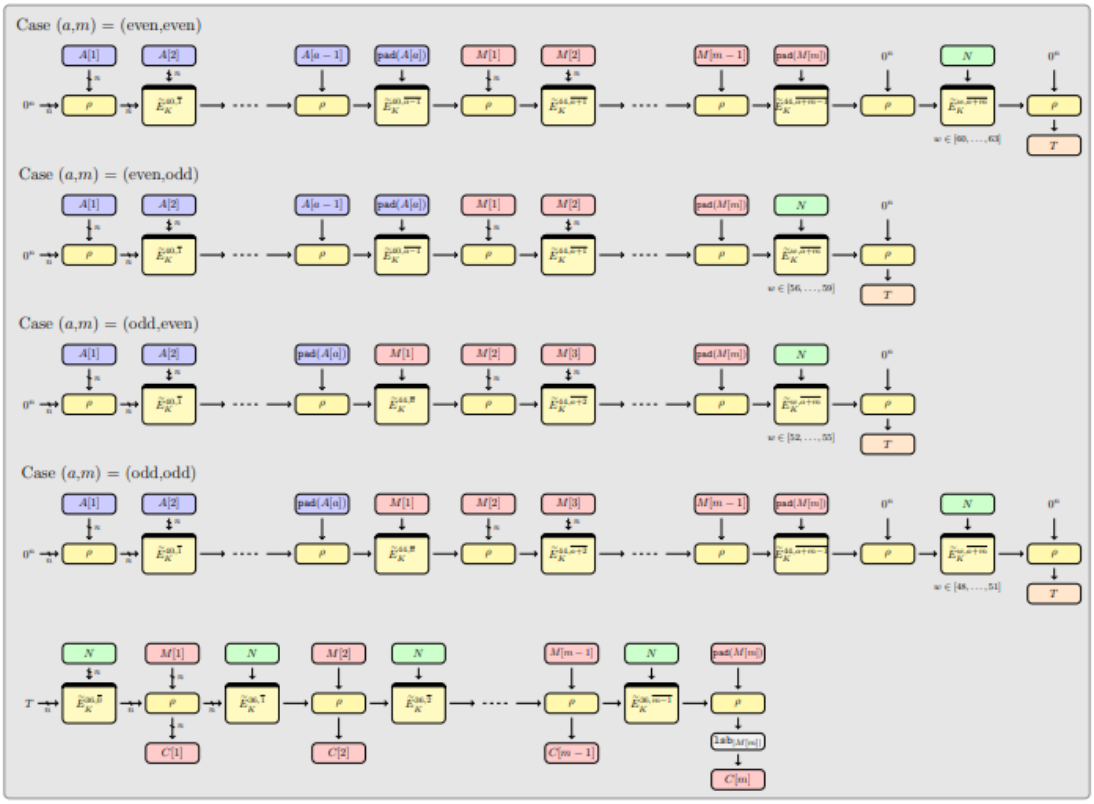
Η υλοποίηση σε υλικό της πιο ελαφριάς παραλλαγής του Romulus που ανέπτυξαν οι ερευνητές, του Romulus-N, χρειάζεται μόνο 6325 GEs. Ο Romulus-M απαιτεί επίσης παρόμοιο μέγεθος περιοχής πύλης (Gate area).

AEAD variants	*Family*	*TBC*	*# Rounds*	*Key*	*Nonce*	*Tag*
Romulus-N1		SKINNY-128-384	56	128	128	128
Romulus-N2	Romulus-N	SKINNY-128-384	56	128	96	128
Romulus-N3		SKINNY-128-256	48	128	96	128
Romulus-M1		SKINNY-128-384	56	128	128	128
Romulus-M2	Romulus-M	SKINNY-128-384	56	128	96	128
Romulus-M3		SKINNY-128-256	48	128	96	128

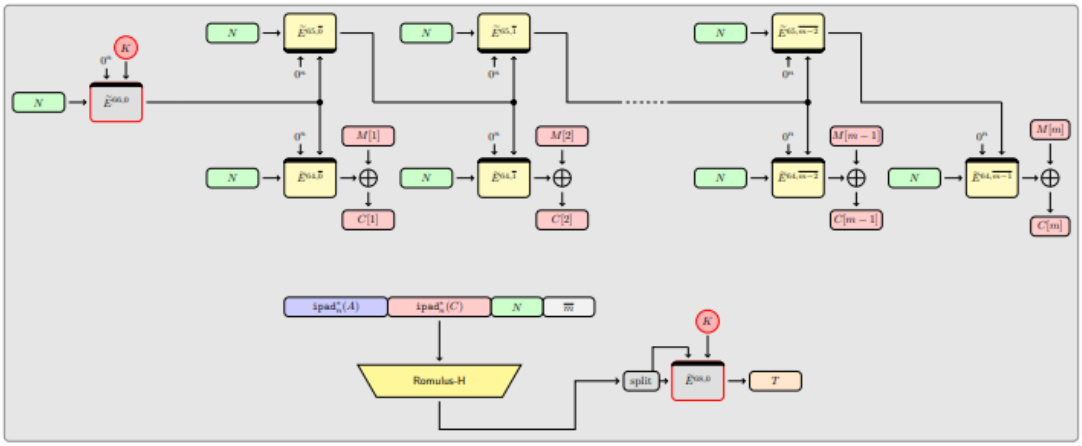
Πίνακας 4.12: Παραλλαγές-Χαρακτηριστικά αλγορίθμου Romulus



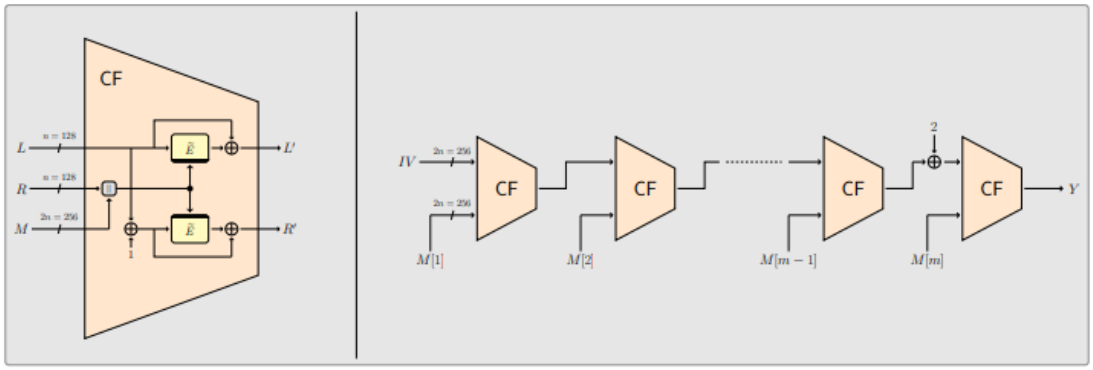
Σχήμα 4.13: Κρυπτογράφηση Romulus N



Σχήμα 4.14: *Κρυπτογράφηση* Romulus *M*



Σχήμα 4.15: *Κρυπτογράφηση* Romulus *T*



Σχήμα 4.16 *: Κρυπτογράφηση Romulus H*

4.8 Αλγόριθμος SPARKLE

Ο SPARKLE είναι μια οικογένεια κρυπτογραφήσεων αντιμετάθεσης που αποτελείται από έναν αυθεντικό κρυπτογράφο που ονομάζεται Sponge-based cipher for Hardened but Weightless Authenticated Encryption on Many Microcontroller (SCHEWAEMM), μια συνάρτηση κατακερματισμού που ονομάζεται Efficient Sponge-based and Cheap Hashing (ESCH). Ο SPARKLE κατασκευάζεται χρησιμοποιώντας μια δομή ARX. Ωστόσο, έχει βελτιωθεί έναντι γραμμικών επιθέσεων. Οι τρεις παραλλαγές του SCHWAEMM που χρησιμοποιούν μεγέθη κλειδιών 128-bit, 256-bit και 192-bit είναι γνωστές ως SCHWAEMM256-128, SCHWAEMM256-256 και SCHWAEMM192-192, αντίστοιχα. Όλα αυτά χρησιμοποιούν μια κατάσταση 256-bit και ένα nonce 256-bit. Το ESCH έχει μέγεθος κατάστασης 384-bit. Η δομή ARX στο SPARKLE εκτελεί XOR και άλλες πράξεις σε λέξεις των 32-bit για να βελτιώσει την απόδοση σε μικροελεγκτές χαμηλού και υψηλού επιπέδου.

Η αντιμετάθεση εφαρμόζει πολλαπλές διαφορετικές περιπτώσεις του Alzette, ενός block cipher 64-bit 4 γύρων, για την επίτευξη της μη γραμμικότητας. Ο Alzette είναι ένας 64-bit S-box που βασίζεται σε ένα Addition-Rotation-XOR

(ARX) το οποίο λειτουργεί σε λέξεις 32-bit, καθιστώντας το ιδιαίτερα αποδοτικό σε ένα λογισμικό. Ο SCHWAEMM, ανήκει στην οικογένεια κρυπτογράφησης AEAD και βασίζεται στην κατασκευή ενός διπλού σφουγγαριού με μια συνδυασμένη ανατροφοδότηση (COFB). Η οικογένεια των συναρτήσεων κατακερματισμού ESCH βασίζεται στην κατασκευή σφουγγαριού.

Η σουίτα SPARKLE αποτελείται από πολλαπλούς αλγορίθμους:

- Ο **Sparkle** είναι μια οικογένεια κρυπτογραφικών μεταθέσεων, κάθε μία από τις οποίες λειτουργεί σε διαφορετικό μέγεθος μπλοκ (256, 384 ή 512 bit). Βασίζονται μόνο στη πρόσθεση, τις περιστροφές (rotations) και XOR. Είναι δυνατόν να γραφτεί μια καθολική υλοποίηση για όλες τις παραλλαγές που λαμβάνει απλώς το μέγεθος του μπλοκ και τον αριθμό των βημάτων ως εισόδους. Βασίζεται στον μετασχηματισμό 64-bit που ονομάσαμε Alzette και τον συμβολίζουμε Ac.
- Ο **Schwaemm** είναι ένας αλγόριθμος AEAD που χρησιμοποιεί τις μεταθέσεις Sparkle σε μια συγκεκριμένη λειτουργία. Υπάρχουν διάφορες εκδόσεις, καθεμία από τις οποίες παρέχει ένα συγκεκριμένο επίπεδο ασφάλειας και απαιτεί μια αντιμετάθεση που λειτουργεί σε ένα συγκεκριμένο μέγεθος μπλοκ.
- Ο **Esch** είναι μια οικογένεια συναρτήσεων κατακερματισμού. Η Esch256 εξάγει σύνοψη 256 bit και η Esch384 384 bit. Βασίζονται σε σφουγγάρι, όπως το σημερινό πρότυπο κατακερματισμού SHA-3.
- Η **XOEsch** είναι μια οικογένεια Extendable Output Functions (XOF) που βασίζεται σε Esch. Μια XOF είναι μια συνάρτηση κατακερματισμού για την οποία το μέγεθος εξόδου μπορεί να είναι αυθαίρετα μεγάλο. Αντίστροφα, μια XOF μπορεί να θεωρηθεί ως ένας κρυπτογράφος ροής με ένα αυθαίρετα μεγάλο nonce/IV.

AEAD variants	*Key*	*Nonce*	*Tag*	*b, r, c*	*# Steps*	**Hash variants**	*Digest size*	*b, r, c*	*# Steps*
SCHWAEMM128-128	128	128	128	256,128,128	7,10	**ESCH256**	256	384,128,256	7,11
SCHWAEMM192-192	192	192	192	384,192,192	7,11	ESCH384	384	512,128,384	8,12
SCHWAEMM256-128	128	256	128	384,256,128	7,11				
SCHWAEMM256-256	256	256	256	512,256,256	8,12				

Πίνακας 4.17:Παραλλαγές του SPARKLE

4.9 Αλγόριθμος TinyJambu

Ο αλγόριθμος TinyJAMBU, είναι ένα σύστημα AEAD που είναι εμπνευσμένο από τον υποψήφιο του τρίτου γύρου του διαγωνισμού CAESAR, τον JAMBU. Το κύριο συστατικό του TinyJAMBU είναι μια αντιμετάθεση με κλειδί 128-bit χωρίς χρονοδιάγραμμα κλειδιών, που βασίζεται σε έναν μη γραμμικό καταχωρητή μετατόπισης ανατροφοδότησης και η μη γραμμικότητα σε κάθε γύρο επιτυγχάνεται με τη χρήση μίας και μόνο λειτουργίας NAND.

Χρησιμοποιεί μικρότερα μεγέθη μπλοκ και μια πιο ελαφριά μετατροπή με κλειδί. Η κύρια παραλλαγή, το TinyJAMBU-128, χρησιμοποιεί ένα κλειδί 128 bit και μια κατάσταση 128 bit. Το κύριο συστατικό της αντιμετάθεσης με κλειδί είναι ένας μη γραμμικός καταχωρητής ανατροφοδότησης (NFRS). Ο NFSR χρησιμοποιείται για την ενημέρωση της τρέχουσας κατάστασης. Επιπλέον, ο αριθμός των γύρων είναι 640 ή 1024, ανάλογα με τη φάση λειτουργίας της κρυπτογράφησης. Το ιδιαίτερο χαρακτηριστικό του NFSR είναι ότι 32 ενημερώσεις μπορούν να εκτελεστούν παράλληλα σε μια CPU 32-bit. Οι αλγόριθμοι κρυπτογράφησης και αποκρυπτογράφησης έχουν τέσσερις κύριες φάσεις, την αρχικοποίηση, την επεξεργασία των σχετικών δεδομένων, την επεξεργασία του απλού κειμένου/κρυπτογραφημένου κειμένου και την οριστικοποίηση/επαλήθευση. Μια ετικέτα 64-bit παράγεται στη φάση οριστικοποίησης του αλγορίθμου κρυπτογράφησης. Οι άλλες δύο παραλλαγές, TinyJAMBU-192 και TinyJAMBU-256, έχουν κλειδιά 192 και 256 bit.

Σύμφωνα με τις προδιαγραφές, το TinyJAMBU-128 έχει ασφάλεια 112-bit, ενώ το TinyJAMBU-192 και το TinyJAMBU-256 έχουν ασφάλεια 168-bit και 224-bit, αντίστοιχα.

Οι παραλλαγές της οικογένειας TinyJAMBU παρατίθενται παρακάτω.

AEAD variants	*Key*	*Nonce*	*Tag*	*State size*
TinyJAMBU-128	128	96	64	128
TinyJAMBU-192	192	96	64	128
TinyJAMBU-256	256	96	64	128

Πίνακας 4.18: Παραλλαγές- Χαρακτηριστικά του TinyJambu**.**

4.10 Αλγόριθμος Xoodyak

Ο αλγόριθμος Xoodyak, είναι ένα σύστημα AEAD και κατακερματισμού με βάση την αντιμετάθεση. Βασίζεται σε μια σταθερή αντιμετάθεση 384-bit Xoodoo που λειτουργεί σε Cyclist mode. Η σχεδιαστική προσέγγιση του Xoodoo είναι στενά συνδεδεμένη με την αντιμετάθεση KECCAK.

Υπάρχουν δύο διαφορετικές λειτουργίες του XOODYAK, η λειτουργία κατακερματισμού και η λειτουργία με κλειδί. Όταν αρχικοποιείται με ένα κλειδί, μπορεί να χρησιμοποιηθεί σε λειτουργία με κλειδί. Ο τρόπος λειτουργίας που

χρησιμοποιείται στον XOODYAK ονομάζεται Cyclist. Το Cyclist περιέχει τη κατάσταση primitive.

AEAD variants	*Key*	*Nonce*	*Tag*	**Hash variants**	*Digest size*
Xoodyakv1	128	128	128	**Xoodyakv1**	256

Πίνακας 4.19:Χαρακτηριστικά του Xoodyak

4.11 Πίνακας χαρακτηριστικών των lightweight αλγορίθμων

Στο πίνακα που ακολουθεί περιγράφονται συνοπτικά ο τύπος του κάθε αλγόριθμου, από τους οποίους περιγράφονται παραπάνω, οι παραλλαγές τους και κάποια κομβικά χαρακτηριστικά, όπως αποτελούν ο τρόπος λειτουργίας, το

κλειδί, ο ρυθμός/μπλοκ, η ετικέτα και η ασφάλεια, όλα μετρήσιμα με την μονάδα των BITS.

Name	Type	Variant	Underlying Primitive	State (Bits)	Key (Bits)	Mode of Operation	Rate/Block (Bits)	Tag (Bits)	Security (Bits)
Ascon	Sponge	Ascon-128	Ascon-p	320	128	Duplex	64	128	128
		Ascon-128a	Ascon-p	320	128	Duplex	128	128	128
Elephant	Sponge	Dumbo	Spongent	160	128	Elephant	160	64	112
		Jumbo	Spongent	176	128	Elephant	176	64	127
		Delirium	Keccak	200	128	Elephant	176	128	127
GIFT-COFB	Block	GIFT-COFB	GIFT-128	192	128	COFB	128	128	128
Grain-128AEAD	Stream	Grain-128AEAD	N/A	256	128	N/A	1	64	128
ISAP	Sponge	ISAP-K-128	Keccak	400	128	ISAP	144	128	128
		ISAP-A-128	Ascon-p	320	128	ISAP	64	128	128
		ISAP-K-128A	Keccak	400	128	ISAP	144	128	128
		ISAP-A-128A	Ascon-p	320	128	ISAP	64	128	128
PHOTON-Beetle	Sponge	PHOTON-Beetle-AEAD[128]	PHOTON256	256	128	Beetle	128	256	121
		PHOTON-Beetle-AEAD[32]	PHOTON256	256	128	Beetle	32	256	128
Romulus	Block	Romulus-N	Skinny-128-384	384	128	COFB	128	128	128
		Romulus-M	Skinny-128-384	384	128	COFB	128	128	128
		Romulus-T	Skinny-128-384	384	128	COFB	128	128	128
SPARKLE	Sponge	SCHWAEMM128-128	SPARKLE	256	128	SPARKLE	128	128	120
		SCHWAEMM256-128	SPARKLE	384	128	SPARKLE	256	128	120
		SCHWAEMM192-192	SPARKLE	384	192	SPARKLE	192	192	184
		SCHWAEMM256-256	SPARKLE	512	256	SPARKLE	256	256	248
TinyJambu	Sponge	TinyJambu	TinyJambu	128	128	TinyJambu	32	64	120
Xoodyak	Sponge	Xoodyak	Xoodoo	384	128	Cyclist	352	128	128

Πίνακας 4.20: Χαρακτηριστικά αλγορίθμων

Κεφάλαιο 5

Πειράματα & Συγκριτική Αξιολόγηση

Για την υλοποίηση των πειραμάτων επιλέχθηκε συσκευή αρκετά χαμηλών πόρων όσο και η γλώσσα προγραμματισμού C, η οποία είναι μία αρκετά γρήγορη γλώσσα αλλά και ελαφριά σε σύγκριση με άλλες και αυτό αποτελεί ένα ισχυρό πλεονέκτημα για περιπτώσεις όπως η κρυπτογράφηση και η αποκρυπτογράφηση .

Ο κώδικας των αλγορίθμων λήφθηκε από τον σύνδεσμο https://csrc.nist.gov/projects/lightweight-cryptography/finalists και λόγω κάποιων ελλιπή αρχείων έγινε pull και από τη πλατφόρμα GitHub (σύνδεσμοι https://github.com/rweather/lwc-finalists, https://github.com/rweather/ascon-suite.

Πέραν από τις μετρήσεις που πραγματοποιήθηκαν στη συγκεκριμένη συσκευή, για την καλύτερη σωστή αξιολόγηση και επιλογή, συλλέχθηκαν και κάποιες μετρήσεις από πειράματα που έγιναν καθ' όλη τη διάρκεια του διαγωνισμού που πραγματοποίησε ο NIST, καθώς έχουν πραγματοποιηθεί σε εύρος lightweight συσκευών το οποίο είναι σημαντικό, διότι σε κάθε συσκευή, η λειτουργία κρυπτογράφησης/αποκρυπτογράφησης κάθε αλγορίθμου, διαφέρει σημαντικά.

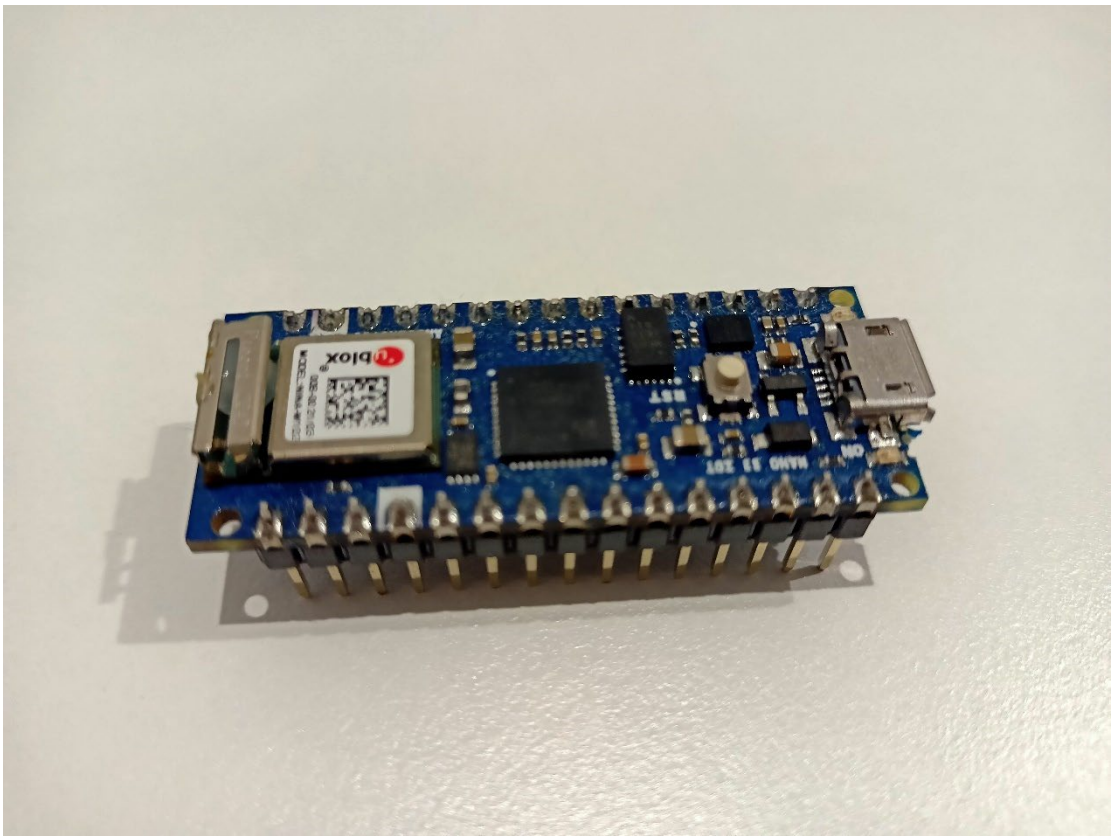
5.1 Υπολογιστικές συσκευές και προγραμματιστικό περιβάλλον

5.1.1 Χαρακτηριστικά ARDUINO NANO 33 IoT

- Μοντέλο: ARDUINO NANO 33
- Chip: ATSAMB21
- Flash memory: 265KB, SRAM 32 KB
- Clock: 48 MHz
- Voltages: 5V
- Συνδεσιμότητα: Η συνδεσιμότητα του επιτυγχάνεται με ένα καλώδιο USB τάσεως 5 VOLT

Το Arduino είναι μια ηλεκτρονική πλατφόρμα ανοικτού κώδικα σχεδιασμού, η οποία στηρίζει τις βάσεις της σε ένα εύκολο και ευέλικτο στη χρήση υλικό και λογισμικό. Έχει δημιουργηθεί για να εξυπηρετήσει και να βοηθήσει καλλιτέχνες, σχεδιαστές και γενικά οποιονδήποτε έχει ενδιαφέρον στο να δημιουργήσει

περιβάλλοντα και αντικείμενα που αλληλοεπιδρούν μεταξύ τους, καθώς και για την υλοποίηση χόμπι και δραστηριοτήτων.



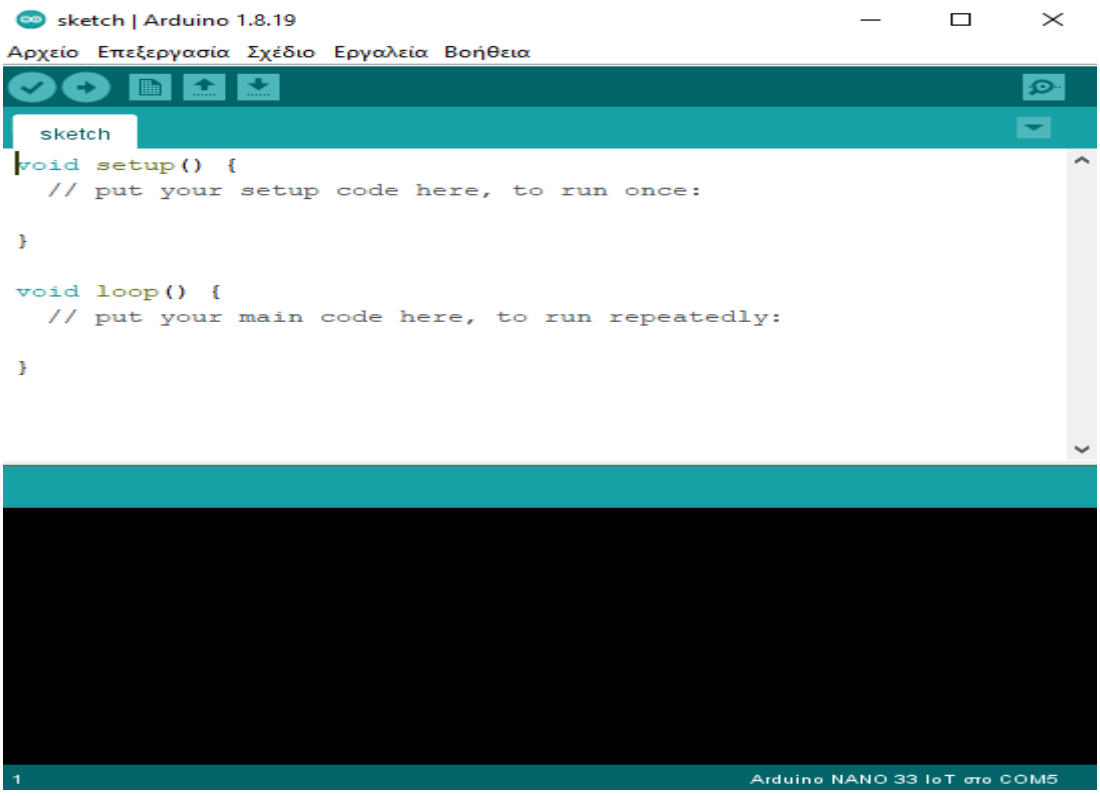
Εικόνα 5.1: Arduino Nano 33 IoT

5.1.2 IDE (Integrated Development Environment):

Το προγραμματιστικό περιβάλλον του ARDUINO ή αλλιώς το περιβάλλον ανάπτυξής, περιλαμβάνει ένα πρόγραμμα επεξεργασίας κειμένου για την σύνταξη του κώδικα, μια περιοχή στην οποία εμφανίζονται τα μηνύματά, μια κονσόλα κειμένου και μια γραμμή εργαλείων.

Η εν λόγω συσκευή είναι αυτή που μας βοήθησε να κάνουμε την ανάλυσή μας καθώς μέσω αυτής τρέξαμε τους αλγόριθμους, βγάλαμε/συγκεντρώσαμε τα αποτελέσματά μας και κάναμε την ανάλυσή μας.

Εικόνα 5.2:Λογισμικό Arduino v1.8.19



Εικόνα 5.3: Περιβάλλον πειραμάτων

5.2 Πειράματα/Μετρήσεις Αλγορίθμων

Ο κώδικα λήφθηκε από το επίσημο site του NIST και τη πλατφόρμα GitHub σε γλώσσα C, όπου με κάποιες μικρές παραμετροποιήσεις και προσθήκες στο φάκελο libraries του Arduino, έγινε δοκιμή για κάθε αλγόριθμο ως προς την απόδοση με αποτέλεσμα τα παρακάτω δεδομένα τα οποία έτρεξαν σε συσκευή χαμηλών πόρων 32bit και 48MHz clock. Το baud rate της σειριακής θύρας ορίστηκε στα 9600 bits per second.

Για τη λήψη των μετρήσεων δημιουργήθηκε sketch, όπου διαμορφώθηκε ανάλογα για τον κάθε αλγόριθμο με βάση τα χαρακτηριστικά του όπως το κλειδί, η ετικέτα (tag) και το nonce (ΠΑΡΑΡΤΗΜΑ Β). Το μέγεθος των κειμένων στα οποία έγινε κρυπτογράφηση/αποκρυπτογράφηση είναι 128 bytes και αντίστοιχα του δεύτερου στα 16 bytes. Επίσης ο κατακερματισμός πραγματοποιήθηκε σε μεγέθη 1024, 128 και 16 bytes.

Η διαδικασία τόσο της κρυπτογράφησης/αποκρυπτογράφησης όσο και του κατακερματισμού έγινε μαζικά για τη σωστή λήψη των αποτελεσμάτων.

5.2.1 Δεδομένα αλγορίθμου ASCON

Στον ASCON και για τις δύο παραλλαγές του, χρησιμοποιήθηκε το ίδιο κλειδί, ετικέτα και nonce σε hex των 128 bits για τη κρυπτογράφηση/αποκρυπτογράφηση σε κείμενο 128 bytes όσο και σε 16 bytes.

- **Key**= 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F
- **Nonce**= 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
- **Tag**= 0x10, 0x11, 0x0A, 0x13, 0x14, 0x0B, 0x16, 0x17, 0x00, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

Αλγόριθμος	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
ASCON											
Ascon-128	128	128	128	44.67	22387.55	45.88	21794.56	220.31	4539.05	222.00	4504.59
Ascon-128a	128	128	128	39.71	25181.54	40.93	24432.99	215.36	4643.43	217.05	4607.27

Πίνακας 5.20: *Δεδομένα Ascon - Enc&Decr*

Επίσης για κατακερματισμό size 256 bits σε 1024 bytes, 128 και 16 bytes.

Παρατηρείται και στις δύο παραλλαγές τα δεδομένα δε διαφέρουν πολύ στα 16 bytes και στα 1024 bytes δίνουν παρόμοια αποτελέσματα.

Hash	Digest size(Bits)	1024 bytes		128 bytes	16 bytes
Ascon	256	6.75		8.33	20.94
Ascon Xof	256	6.75		8.32	20.87
Ascon Hasha	256	4.8		4.30	18.21
Ascon Xofa	256	4.8		4.29	18.18

Πίνακας 5.21: *Δεδομένα Ascon - Hash*

5.2.2 Δεδομένα αλγορίθμου Elephant

Στον Elephant και για τις τρείς παραλλαγές του, χρησιμοποιήθηκε κλειδί και nonce σε hex των 128 bits key, 96 bits nonce και ετικέτα 64 bits για τη Dumbo και Jumbo ενώ για τη Delirium έκδοση 128 bits ετικέτα για κρυπτογράφηση/αποκρυπτογράφηση σε κείμενο 128 bytes όσο και σε 16 bytes.

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
- **Nonce=**0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B
- **Tag**= 0x1E, 0x0A, 0x13, 0x00, 0x1A, 0x1B, 0x1C, 0x1D

Αλγόριθμος	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
ELEPHANT											
Dumbo	128	64	96	296.46	34192.22	292.71	3416.32	583.41	1713.90	584.13	1711.94
Jumbo	128	64	96	314.86	3175.97	315.13	3173.28	718.57	1391.65	719.24	1390.35
Delirium	128	128	96	29.77	33592.30	28.97	33231.45	66.09	15147.20	66.83	14963.90

Πίνακας 5.22: *Δεδομένα Elephant - Enc&Decr*

Επίσης για κατακερματισμό size 256 bits και αντίστοιχα 384 σε μέγεθος 1024 bytes, 128 και 16 bytes, να παρατηρείται σχεδόν διπλάσια διαφορά στα αποτελέσματα στα 16 bytes μεταξύ τους.

Hash	Digest size(Bits)	1024 bytes		128 bytes	16 bytes
Elephant					
Esch256	256	4.83		5.67	12.46
Esch384	384	7.06		8.96	24.11

Πίνακας 5.23: *Δεδομένα Elephant - Hash*

5.2.3 Δεδομένα αλγορίθμου GIFT-COFB

Στον Gift-Cofb, χρησιμοποιήθηκε κλειδί, nonce και tag σε hex των 128 bits για τη κρυπτογράφηση/αποκρυπτογράφηση σε κείμενο 128 bytes όσο και σε 16 bytes.

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
- **Nonce=**0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x0C, 0x0D, 0x0E, 0x0B
- **Tag**= 0x10, 0x11, 0x0A, 0x13, 0x14, 0x0B, 0x16, 0x17, 0x00, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

Αλγόριθμος	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
GIFT-COFB											
GIFT-128	128	128	128	5.34	187193.34	5.59	178892.88	15.54	64362.06	16.11	62064.18

Πίνακας 5.24: *Δεδομένα Gift-Cofb - Enc&Decr*

Παρατηρείται κατά την κρυπτογράφηση/αποκρυπτογράφηση στα 128 bytes να έχει καλύτερα αποτελέσματα άνω του διπλάσιου απ' ότι στα 16 bytes.

5.2.4 Δεδομένα αλγορίθμου Grain-128AEAD

Στον αλγόριθμο Grain-128 AEAD χρησιμοποιήθηκε κλειδί, nonce και tag σε hex των 128 bits key, 64 bits tag και 96 bits nonce για τη κρυπτογράφηση/αποκρυπτογράφηση σε κείμενο 128 bytes και σε 16 bytes με καλύτερα αποτελέσματα σε plaintext των 128 bytes.

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
- **Nonce=**0x10, 0x11, 0x0B, 0x13, 0x14, 0x15, 0x18, 0x19, 0x1A, 0x1B, 0x0C, 0x0D
- **Tag**= 0x1E, 0x0A, 0x13, 0x00, 0x1A, 0x1B, 0x1C, 0x1D

Αλγόριθμος	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
Grain-128AEAD											
Grain-128AEAD	128	64	96	20.63	48463.79	20.86	47946.83	32.17	31080.49	32.46	30804.91

Πίνακας 5.25: *Δεδομένα Grain-128AEAD - Enc&Decr*

5.2.5 Δεδομένα αλγορίθμου ISAP

Στον αλγόριθμο ISAP σε όλες τις παραλλαγές, χρησιμοποιήθηκε κλειδί, nonce και tag σε hex των 128 bits για κρυπτογράφηση/αποκρυπτογράφηση σε κείμενο 128 bytes και σε 16 bytes αντίστοιχα.

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F

- **Nonce=**0x10, 0x11, 0x0B, 0x13, 0x14, 0x15, 0x18, 0x19, 0x1A, 0x1B, 0x0C, 0x0D, 0x00, 0x09, 0x0A, 0x05
- **Tag=** 0x10, 0x11, 0x0A, 0x13, 0x14, 0x0B, 0x16, 0x17, 0x00, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

Αλγόριθμος	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
ISAP											
ISAP-K-128	128	128	128	841.32 -	1188.61	6286.22 -	159.08	841.60 -	1188.22	6286.81 -	159.06
ISAP-A-128	128	128	128	174.58 -	5728.05	174.86 -	5718.93	1258.13 -	794.83	1258.73 -	794.45
ISAP-K-128A	128	128	128	132.63 -	7539.68	132.91 -	7523.88	726.75 -	1376.00	727.35 -	1374.85
ISAP-A-128A	128	128	128	38.49 -	25979.35	38.77 -	25793.24	201.06 -	4973.71	201.66 -	4958.78

Πίνακας 5.26: *Δεδομένα ISAP- Enc&Decr*

Όπως παρατηρείται, κατά την κρυπτογράφηση/αποκρυπτογράφηση και στα δύο μεγέθη 128 και 16 bytes η παραλλαγή ISAP-A-128A να έχει διαφορά στα αποτελέσματα σε σχέση με τις υπόλοιπες παραλλαγές.

5.2.6 Δεδομένα αλγορίθμου PHOTON-Beetle

Στον αλγόριθμο Photon-Beetle και στις δύο παραλλαγές του, χρησιμοποιήθηκε το ίδιο κλειδί και nonce σε hex των 128 bits και ετικέτα 256 bits για τη κρυπτογράφηση/αποκρυπτογράφηση σε 128 bytes όσο και σε 16 bytes plaintext.

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
- **Nonce=**0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x0C, 0x0D, 0x03, 0x0B
- **Tag=** 0x10, 0x11, 0x0A, 0x13, 0x14, 0x0B, 0x06, 0x07, 0x16, 0x17, 0x00, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x0E, 0x0F, 0x0C, 0x0D, 0x03, 0x17, 0x18, 0x19, 0x2A, 0x1B, 0x2C, 0x2D, 0x1E, 0x2F

Αλγόριθμος	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
PHOTON-Beetle											
PHOTON-Beetle AEAD[128	128	256	128	25.79 -	38779.31	26.62 -	38425.94	45.82 -	21826.79	46.18 -	21656.14
PHOTON-Beetle-AEAD[32]	128	256	128	92.05 -	10863.82	92.31 -	10833.15	112.12 -	8919.14	42.50 -	8889.23

Πίνακας 5.27: *Δεδομένα Photon Beetle – Enc & Decr*

Για κατακερματισμό size 256 bits σε μέγεθος 1024, 128 και 16 bytes, να παρατηρείται μεγάλη διαφορά στα αποτελέσματα των 1024 και 128 bytes από τα 16 bytes.

Hash	Digest size(Bits)	1024 bytes		128 bytes	16 bytes
Photon-Beetle-Hash[32]	256	87.74		83.00	45.03

Πίνακας 5.28: *Δεδομένα Photon Beetle - Hash*

5.2.7 Δεδομένα αλγορίθμου Romulus

Στον αλγόριθμο Romulus σε όλες τις εκδόσεις του, χρησιμοποιήθηκε κλειδί, nonce και ετικέτα σε hex των 128 bits για τη κρυπτογράφηση/αποκρυπτογράφηση σε 128 bytes όσο και σε 16 bytes plaintext.

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
- **Nonce=**0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x0C, 0x0D, 0x03, 0x0B
- **Tag**= 0x10, 0x11, 0x0A, 0x13, 0x14, 0x0B, 0x16, 0x17, 0x00, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

Αλγόριθμος	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
Romulus											
Romulus-N	128	128		26.94	37113.07	27.17	36803.11	62.28	14435.21	69.84	14318.49
Romulus-M	128	128		44.09	22680.73	44.31	22566.95	103.45	9666.67	104.00	9615.66
Romulus-T	128	128		104.87	9535.80	105.14	9511.29	235.38	4248.45	235.99	4237.54

Πίνακας 5.28: *Δεδομένα Romulus– Enc & Decr*

Για κατακερματισμό size 256 bits σε μέγεθος 1024, 128 και 16 bytes, να παρατηρείται μεγάλη διαφορά στα αποτελέσματα 128 bytes από τα άλλα μεγέθη.

Hash	Digest size(Bits)	1024 bytes	128 bytes	16 bytes
Romulus	256	28.70	84.93	57.05

Πίνακας 5.30: *Δεδομένα Romulus- Hash*

5.2.8 Δεδομένα αλγορίθμου SPARKLE

Στον αλγόριθμο SPARKLE σε κάθε μία έκδοση διαφέρει το κλειδί η ετικέτα αλλά και το nonce μεταξύ τους. Στην έκδοση SCHWAEMM128-128 το key, nonce και η ετικέτα είναι 128 bits, στη SCHWAEMM192-192 κάνει χρήση 192 bits, στη SCHWAEMM256-128 το key και η ετικέτα είναι 128 bits ενώ το nonce 256 και στην τέταρτη παραλλαγή SCHWAEMM256-256 το key, nonce και η ετικέτα είναι 256 bits για κρυπτογράφηση/αποκρυπτογράφηση σε plaintext 128 bytes όσο και σε 16 bytes.

SCHWAEMM128-128

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F

- **Nonce=**0x10, 0x11, 0x0B, 0x13, 0x14, 0x15, 0x18, 0x19, 0x1A, 0x1B, 0x0C, 0x0D, 0x00, 0x09, 0x0A, 0x05
- **Tag**= 0x10, 0x11, 0x0A, 0x13, 0x14, 0x0B, 0x16, 0x17, 0x00, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

SCHWAEMM192-192

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17
- **Nonce=**0x10, 0x11, 0x0B, 0x13, 0x14, 0x15, 0x18, 0x19, 0x1A, 0x1B, 0x0C, 0x0D, 0x00, 0x09, 0x0A, 0x05, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F
- **Tag**= 0x0A, 0x13, 0x14, 0x0B, 0x06, 0x07, 0x16, 0x17, 0x00, 0x19, 0x1A,0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x0E, 0x0F, 0x0C, 0x0D, 0x03, 0x17, 0x18,0x03

SCHWAEMM256-128

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
- **Nonce=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F
- **Tag**= 0x10, 0x11, 0x0A, 0x13, 0x14, 0x0B, 0x16, 0x17, 0x00, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

SCHWAEMM256-256

- **Key=** 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F
- **Nonce =**0x00, 0x1C, 0x01, 0x05, 0x06, 0x07,0x08, 0x11, 0x12, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1D, 0x1E, 0x1F, 0x02, 0x03, 0x04
- **Tag**= 0x10, 0x11, 0x0A, 0x13, 0x14, 0x0B, 0x06, 0x07, 0x16, 0x17, 0x00, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x0E, 0x0F, 0x0C, 0x0D, 0x03, 0x17, 0x18, 0x19, 0x2A, 0x1B, 0x2C, 0x2D, 0x1E, 0x2F

Αλγόριθμος	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
SPARKLE											
SCHWAEMM128-128	128	128	128	4.46	224284.30	4.56	219319.29	9.92	100777.88	10.32	26869.31
SCHWAEMM192-192	192	192	192	5.34	187118.36	5.40	185138.58	15.68	63769.33	16.11	62069.88
SCHWAEMM256-128	128	128	256	4.22	237035.72	4.35	229861.11	15.55	64319.11	15.91	62859.07
SCHWAEMM256-256	256	256	256	5.75	173870.05	5.90	169492.42	21.57	46353.8	22.3	44783.76

Πίνακας 5.31: *Δεδομένα SPARKLE– Enc & Decr*

5.2.9 Δεδομένα αλγορίθμου Tiny-Jambu

Ο αλγόριθμος Tiny JAMBU σε όλες τις εκδόσεις του, χρησιμοποιεί διαφορετικό μέγεθος κλειδιού όπου στη παραλλαγή TinyJambu128 128 bits key, στη TinyJambu192 192 bits key και στην TinyJambu256 192 bits key. Σε όλες τις παραλλαγές τα nonce και ετικέτα σε hex είναι των 96 bits nonce και 64 bits tag για κρυπτογράφηση/αποκρυπτογράφηση σε 128 bytes όσο και σε 16 bytes plaintext.

TinyJambu128

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F

TinyJambu192

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17

TinyJambu256

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

Nonce και **Tag** για όλες τις εκδόσεις:

- **Nonce=**0x10, 0x11, 0x0B, 0x13, 0x14, 0x15, 0x18, 0x19, 0x1A, 0x1B, 0x0C, 0x0D
- **Tag=**0x10, 0x11, 0x0B, 0x1A, 0x1B, 0x0C, 0x0D, 0x0A

Αλγόριθμος	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
TinyJambu											
TinyJambu128	128	64	96	6.34	157837.04	6.62	151100.73	12.08	82782.74	12.57	79532.61
TinyJambu192	192	64	96	6.91	144699.15	7.19	139025.86	12.90	77548.69	13.39	74710.26
TinyJambu256	256	64	96	7.40	135051.01	7.69	130088.31	13.57	73677.15	14.07	71091.52

Πίνακας 5.32: *Δεδομένα Tiny-Jambu – Enc & Decr*

5.2.10 Δεδομένα αλγορίθμου Xoodyak

Ο αλγόριθμος Xoodyak χρησιμοποιεί κλειδί, nonce και tag σε hex των 128 bits για τη κρυπτογράφηση/αποκρυπτογράφηση σε 128 bytes όσο και σε 16 bytes plaintext.

- **Key=**0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F

- **Nonce=**0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x0C, 0x0D, 0x03, 0x0B
- **Tag=** 0x10, 0x11, 0x0A, 0x13, 0x14, 0x0B, 0x16, 0x17, 0x00, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

Αλγόριθμος	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
Xoodyak											
Xoodyak	128	128	128	5.04	198371.49	5.25	190327.21	15.08	66318.13	15.53	64372.42

Πίνακας 5.33: *Δεδομένα Xoodyak – Enc & Decr*

Για κατακερματισμό size 256 bits σε μέγεθος 1024 bytes, 128 και 16 bytes, παρατηρείται τριπλάσια διαφορά στα αποτελέσματα των 16 bytes.

Hash	Digest size(Bits)	1024 bytes		128 bytes	16 bytes
Xoodyak	256	5.05		5.99	16.07

Πίνακας 5.34: *Δεδομένα Xoodyak – Hash*

5.3 Συγκριτική Αξιολόγηση

Παρακάτω ακολουθούν οι πίνακες με όλα τα αποτελέσματα τις κάθε μία έκδοσης που πραγματοποιήθηκαν στα πειράματα μας.

Αλγόριθμοι	Key (bits)	Tag(bits)	Nonce(bits)	Κρυπτογράφηση 128 bytes		Αποκρυπτογράφηση 128 bytes		Κρυπτογράφηση 16 bytes		Αποκρυπτογράφηση 16 bytes	
				cycles per byte	*bytes per sec*	cycles per byte	bytes per sec	cycles per byte	bytes per sec	cycles per byte	bytes per sec
ASCON											
Ascon-128	128	128	128	44.67	22387.55	45.88	21794.56	220.31	4539.05	222.00	4504.59
Ascon-128a	128	128	128	39.71	25181.54	40.93	24432.99	215.36	4643.43	217.05	4607.27
ELEPHANT											
Dumbo	128	64	96	296.46	34192.22	292.71	3416.32	583.41	1713.90	584.13	1711.94
Jumbo	128	64	96	314.86	3175.97	315.13	3173.28	718.57	1391.65	719.24	1390.35
Delirium	128	128	96	29.77	33592.30	28.97	33231.45	66.09	15147.20	66.83	14963.90
GIFT-COFB											
GIFT-128	128	128	128	5.34	187193.34	5.59	178892.88	15.54	64362.06	16.11	62064.18
Grain-128AEAD											
Grain-128AEAD	128	64	96	20.63	48463.79	20.86	47946.83	32.17	31080.49	32.46	30804.91
ISAP											
ISAP-K-128	128	128	128	841.32	1188.61	6286.22	159.08	841.60	1188.22	6286.81	159.06
ISAP-A-128	128	128	128	174.58	5728.05	174.86	5718.93	1258.13	794.83	1258.73	794.45
ISAP-K-128A	128	128	128	132.63	7539.68	132.91	7523.88	726.75	1376.00	727.35	1374.85
ISAP-A-128A	128	128	128	38.49	25979.35	38.77	25793.24	201.06	4973.71	201.66	4958.78

Πίνακας 5.20: *Πίνακας δεδομένων 1 - Enc&Decr*

PHOTON-Beetle							
PHOTON-Beetle AEAD[128	128	256	128	25.79 - 38779.31	26.62 - 38425.94	45.82 - 21826.79	46.18 - 21656.14
PHOTON-Beetle-AEAD[32]	128	256	128	92.05 - 10863.82	92.31 - 10833.15	112.12 - 8919.14	42.50 - 8889.23
Romulus							
Romulus-N	128	128		26.94 - 37113.07	27.17 - 36803.11	62.28 - 14435.21	69.84 - 14318.49
Romulus-M	128	128		44.09 - 22680.73	44.31 - 22566.95	103.45 - 9666.67	104.00 - 9615.66
Romulus-T	128	128		104.87 - 9535.80	105.14 - 9511.29	235.38 - 4248.45	235.99 - 4237.54
SPARKLE							
SCHWAEMM128-128	128	128	128	4.46 - 224284.30	4.56 - 219319.29	9.92 - 100777.88	10.32 - 26869.31
SCHWAEMM192-192	192	192	192	5.34 - 187118.36	5.40 - 185138.58	15.68 - 63769.33	16.11 - 62069.88
SCHWAEMM256-128	128	128	256	4.22 - 237035.72	4.35 - 229861.11	15.55 - 64319.11	15.91 - 62859.07
SCHWAEMM256-256	256	256	256	5.75 - 173870.05	5.90 - 169492.42	21.57 - 46353.8	22.3 - 44783.76
TinyJambu							
TinyJambu128	128	64	96	6.34 - 157837.04	6.62 - 151100.73	12.08 - 82782.74	12.57 - 79532.61
TinyJambu192	192	64	96	6.91 - 144699.15	7.19 - 139025.86	12.90 - 77548.69	13.39 - 74710.26
TinyJambu256	256	64	96	7.40 - 135051.01	7.69 - 130088.31	13.57 - 73677.15	14.07 - 71091.52
Xoodyak							
Xoodyak	128	128	128	5.04 - 198371.49	5.25 - 190327.21	15.08 - 66318.13	15.53 - 64372.42

Πίνακας 5.21: *Πίνακας δεδομένων 2- Enc&Decr*

Hash Αλγορίθμων	Digest size(Bits)	1024 bytes	128 bytes	16 bytes
Ascon	256	6.75	8.33	20.94
Ascon Xof	256	6.75	8.32	20.87
Ascon Hasha	256	4.8	4.30	18.21
Ascon Xofa	256	4.8	4.29	18.18
Photon-Beetle-Hash[32]	256	87.74	83.00	45.03
Romulus	256	28.70	84.93	57.05
Xoodyak	256	5.05	5.99	16.07
Elephant				
Esch256	256	4.83	5.67	12.46
Esch384	384	7.06	8.96	24.11

Πίνακας 5.22: *Πίνακας δεδομένων 3 -Hash*

Οι ανωτέρω μετρήσεις είναι σε bytes per second (B/s) και cycles per byte (c/B) όπου στα δεδομένα φαίνονται κάποιες από τις παραλλαγές των αλγορίθμων να ξεχωρίζουν σημαντικά στη κρυπτογράφηση/αποκρυπτογράφηση τόσο σε 128 bytes όσο και σε 16 αλλά και κάποιες όπως ο Ascon φαίνεται να αποδίδει καλύτερα σε μικρότερο μέγεθος δεδομένων.

Στο κατακερματισμό επίσης ξεχωρίζουν αρκετά ο Romulus και ο Photon Beetle ειδικά σε 128 bytes δεδομένων και οι δύο αλλά στα 1024 bytes να υπερέχει μόνο ο Photon Beetle σε σχέση με τους υπόλοιπους αλγορίθμους.

Κεφάλαιο 6

Επίλογος/Συμπέρασμα

Η ανάλυση των ανωτέρω δέκα αλγορίθμων που έφτασαν έως και τη τελική φάση του διαγωνισμού που διεξήγαγε ο NIST, έγινε τόσο θεωρητικά όσο και πρακτικά. Καθ' όλη τη διάρκεια της έρευνας υπήρχε παρακολούθηση στο επίσημο φόρουμ (https://groups.google.com/a/list.nist.gov/g/lwc-forum) λαμβάνοντας υπόψη σημαντικές πληροφορίες από διάφορα μέλη που εκτέλεσαν σε διάφορες πλατφόρμες πειράματα όπως καθηγητές και τους υπεύθυνους του κάθε αλγορίθμου.

Συγκρίνοντας τα δεδομένα από τα πειράματα στο Arduino όσο και τα δεδομένα που διεξήχθησαν σε άλλες πλατφόρμες όπως ATmega2560 (https://rweather.github.io/lwc-finalists/performance_avr.html), φαίνεται πολλοί από τους αλγορίθμους να διαφέρουν σημαντικά στο θέμα της απόδοσης σε κάθε συσκευή και κυρίως σε κάποια με πολύ περιορισμένους πόρους η οποία θα εκτελεί άνω της μία διεργασίας. Ένας που διαφέρει αρκετά είναι η μία παραλλαγή του Sparkle, Schwaemm128-128. Συμπεραίνουμε επίσης ότι ο Ascon αν και νικητήριος αλγόριθμος προς προτυποποίηση σε συσκευές όπως προαναφερθήκαμε με βάση τα αποτελέσματα ως προς την απόδοση και την όλη ανάλυση θεωρητικά και πρακτικά, δε θα μπορούσε να ανταπεξέλθει σε ικανοποιητικό επίπεδο, πόσο μάλλον αν και ασφαλής, μελλοντικά σε προσθήκη του στο πρωτόκολλο TLS για χρήση πάνω σε συσκευές χαμηλών πόρων, λόγω και τις βαρύτητας της διαδικασίας κρυπτογράφησης/αποκρυπτογράφησης που διεξάγει μία σουίτα όσο και στη μαζική διαδικασία λόγω της επιβάρυνσης σε πόρους .

Βιβλιογραφία

1. Guo C., Iwata T., Khairallah M., Minematsu K., Peyrin T., (Μάιος 2021), Romulus, v1.3, pp. 57. https://romulusae.gihub.io/romulus/
2. Beirle C., Biryukov A., Cardoso dos Santos L., Großschadl J., Moradi A., Perrin L., Rezaei Shahmirzadi A., Udovenko A., Velichkov V., Wang Q.,

(Μάιος 2021), Schwaemm and Esch: Lightweight Authenticated Encryption and Hashing using the Sparkle Permutation Family, v1.2.
3. Fotovvat A., Rahman G. M. E., Graduate Student Member, IEEE, VedaeiS.S, .Wahid K. A., Senior Member, IEEE Internet of Things Journal, vol8, No10, (Μάιος 2021), pp. 8279–90. Comparative Performance Analysis of Lightweight Cryptography Algorithms for IoT Sensor Nodes. https://doi.org/10.1109/JIOT.2020.3044526
4. Gross H., Wenger E., Dobraunig C., Ehrenhöfer C., vol 95, (Νοέμβριος 2016), pp. 470-479, Microprocessors and Microsystems: Ascon hardware implementations and side-channel evaluation.
5. Bundesamt für Sicherheit in der Informations technik, v1, (Ιανουάριος 2022), Kryptographische Verfahren: Empfehlungen und Schlüssellängen, Teil 2 – Verwendung von Transport Layer Security (TLS)
6. Guo C., Pereira O., Peters T, Standaert F.X., (Μάιος 2020), pp. 6-42, Towards Low-Energy Leakage-Resistant Authenticated Encryption from the Duplex Sponge Construction: UCL Crypto Group, Universite catholique de Louvain, Louvain-la-Neuve, Belgium. https://doi.org/10.46586/tosc.v2020.i1.6-42, ISSN: 2519-173X
7. Cheng H., Großschädl J., Marshall B., Page D., Pham T., (Μάιος 2022), pp. 1-18, RISC-V Instruction Set Extensions for Lightweight Symmetric Cryptography: Department of Computer Science, University of Luxembourg, Department of Computer Science, University of Bristol.
8. https://csrc.nist.rip/library
9. Katagi M. and Moriai S., pp. 1-4, Lightweight Cryptography for the Internet of Things
10. Εικόνα2.1,https://subscription.packtpub.com/book/programming/9781838986698/10/ch10lvl1sec72/symmetric-key-encryption
11. https://en.wikipedia.org/wiki/Public-key_cryptography
12. Daemen J., Hoffert S., Peeters M., Gilles V. A., Keer V. R., (Ιούνιος 2020), pp. 60-87, Xoodyak, a lightweight cryptographic scheme: IACR Transactions on Symmetric Cryptology, STMicroelectronics, Diegem, Belgium and Radboud University, Nijmegen, The Netherlands https://tosc.iacr.org/index.php/ToSC/article/view/8618 , DOI: 10.46586/tosc.v2020.iS1.60-87, ISSN: 2519-173X
13. Dmitry Khovratovich, (Ιούλιος 2011), pp. 1-102, Methods of Symmetric Cryptanalysis: Microsoft Research Redmond, USA
14. Singh P., Bibhudendra A., Chaurasiya R. K., (Ιανουάριος 2021), pp. 153-185, Lightweight cryptographic algorithms for resource-constrained IoT devices and sensor networks, In book: Security and Privacy Issues in IoT Devices and Sensor Networks, DOI:10.1016/B978-0-12-821255-4.00008-0

15. Lavaud A. D., Fournet C., Kohlweiss M., Protzenko J., Rastogi A., Swamy N., Béguelin S. Z., Microsoft Research, Bhargavan K., Jianyang P., Zinzindohoué J. K. , INRIA Paris-Rocquencourt, (Μάιος 2017), vol1.3, pp 463-482, Implementing and Proving the TLS: 2017 IEEE Symposium on Security and Privacy, http://ieeexplore.ieee.org/document/7958593/, DOI: 10.1109/SP.2017.58, ISBN: 978-1-5090-5533-3
16. Bellare M. and Tackmann B. (Ιούλιος 2016), pp.247-276, The Multi-user Security of Authenticated Encryption: AES-GCM in TLS 1.3
17. Bhargavan K., Blanchet B., Kobeissi N., (Μάιος 2017), pp. 483-502, Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate: 2017 IEEE Symposium on Security and Privacy (SP), http://ieeexplore.ieee.org/document/7958594/, DOI: 10.1109/SP.2017.26, ISBN: 978-1-5090-5533-3
18. Dowling B, Fischlin M., Felix G., Stebila D., (Οκτώβριος 2021), pp.1-37, A Cryptographic Analysis of the TLS: Journal of Cryptology, vol4(34), https://link.springer.com/10.1007/s00145-021-09384-1, DOI: 10.1007/s00145-021-09384-1, ISSN: 0933-2790, 1432-1378
19. Tange K., Howard D., Shanahan T., Pepe S., Fafoutis X., Dragoni N. , (Μάιος 2017), pp. 243-258, rTLS: Lightweight TLS Session Resumption for Constrained IoT Devices: International Conference on Information and Communications Security, DTU Compute, Technical University of Denmark, Itron Idea Labs, USA UniquID, USA, AASS, Orebro University, https://link.springer.com/10.1007/978-3-030-61078-4_14, DOI: 10.1007/978-3-030-61078-4_14, ISBN: 978-1-5090-5533-3
20. Iosifidis E., Limniotis K. (Νοέμβριος 2016), vol64, pp. 1-5, A study of lightweight block ciphers in TLS: The case of Speck: Proceedings of the 20th Pan-Hellenic Conference on Informatic, DOI: https://doi.org/10.1145/3003733.3003794
21. Bellare M. and Tackmann B., (2016), pp. 247-276, The Multi-user Security of Authenticated Encryption: AES-GCM in TLS 1.3 : Springer Berlin Heidelberg , Department of Computer Science and Engineering, University of California San Diego, USA, http://link.springer.com/10.1007/978-3-662-53018-4_10, ISBN: 978-3-662-53017-7 978-3-662-53018-4
22. Bellare M., Tackmann B., (Νοέμβριος 2017), pp. 247-276, The Multi-User Security of Authenticated Encryption: AES-GCM in TLS 1.3: Advances in Cryptology – CRYPTO 2016, http://link.springer.com/10.1007/978-3-662-53018-4_10, ISBN: 978-3-662-53017-7 978-3-662-53018-4
23. Chiriţă M., Stroie A. M., Safta A. D, Simion E., (2021), pp 1-24, A Note on Advanced Encryption Standard with Galois/Counter Mode Algorithm, Improvements and S-Box Customization.
24. Rescorla E., (Αύγουστος 2018), pp. 1-160, The Transport Layer Security (TLS) Protocol Version 1.3, https://www.rfc-editor.org/info/rfc8446, ISSN: 2070-1721
25. Rescorla E., (Αύγουστος 2018), A Readable Specification of TLS 1.3
26. Nir Y., Salz R., Sullivan N., (Νοέμβριος 2022), Transport Layer Security (TLS) Parameters: Internet Assigned Numbers Authority

27. Protected Hardware Implementations of LWC Finalists:
28. ISAP: Institute of Applied Information Processing and Communications (IAIK), Graz University of Technology, Austria. Ascon, Elephant, GIFT-COFB, ISAP, PHOTON-Beetle, Romulus, SPARKLE, TinyJAMBU, Xoodyak: Ruhr-University Bochum, Germany. Elephant, TinyJAMBU, Xoodyak: Cryptographic Engineering Research Group (CERG), George Mason University, USA. Ascon: Institute of Applied Information Processing and Communications (IAIK), Graz University of Technology, Austria. Xoodyak: Hardware Security and Cryptographic Processor Lab, Tsinghua University, Beijing, China. HPC2: Hardware Private Circuits 2, DOM: Domain-Oriented Masking, GPL-3.0: GNU General Public License version 3
29. Inoue A., Iwata T., Minematsu K., pp.1-18, Analyzing the Provable Security Bounds of
30. GIFT-COFB and Photon-Beetle: NEC Kawasaki Japan - Nagoya University Nagoya Japan
31. Journal of Discrete Mathematical Sciences and Cryptography: A survey on implementation of lightweight block ciphers for resource constraints devices: Nayancy, Sandip Dutta &Soubhik Chakraborty, (Ιανουάριος 2020), vol25 (5), pp. 1377-1398, https://doi.org/10.1080/09720502.2020.1766764
32. IETE Journal of Education: A Review on Evolution of Symmetric Key Block Ciphers and Their Applications: Appala Naidu Tentu (Ιανουάριος 2020), vol61 (1), pp. 34-46 , https://doi.org/10.1080/09747338.2020.1769508
33. Encrypting Private Data In Hacking the Code: Employing Symmetric Cryptography, (2004). Johnson L., Security component fundamentals for assessment: Security Controls Evaluation, Testing, and Assessment Handbook, (2020), vol2. Jason Andress, Cryptography: The Basics of Information Security, (2014), vol2. Fujdiak R., Mlynek P., Security in low-power wide-area networks: state-of-the-art and development toward the 5G: LPWAN Technologies for IoT and M2M Applications, (2020). Kiennert C., Thoniel P., Authentication Systems: Digital Identity Management, (2015). Johnson L., Security Component Fundamentals for Assessment: Security Controls Evaluation, Testing, and Assessment Handbook, (2016). Virtue T., Rainey J., Privacy and Security in Healthcare: HCISPP Study Guide, (2015). Cryptography: In Hack Proofing Your Network (2002), vol2. Timothy J. Shimeall, Jonathan M. Spring, Resistance Strategies: Introduction to Information Security, (2014). Introduction to Information Security, (2014). Hbner S. F., Berthold S., Privacy-Enhancing Technologies: Computer and Information Security Handbook, vol3, (2017)
34. Gros H., Wenger E., Dobrauni C., Ehrenhöfer C., (Ιούλιος 2017), pp. 470-479, Ascon hardware implementations and side-channel evaluation: Microprocessors and Microsystems, https://doi.org/10.1016/j.micpro.2016.10.006
35. Turan M. S., McKay K., Chang D., Calık C., Bassham L., Kang J., Kelsey J., (Ιούλιος 2021), pp.1-92, Status Report on the Second Round of the NIST Lightweight Cryptography Standardization Process: National Institute of

Standards and Technology, https://nvlpubs.nist.gov/nistpubs/ir/2021/NIST.IR.8369.pdf, DOI: 10.6028/NIST.IR.8369
36. Dobraunig C., Eichlseder M., Mendel F., Schläffer M., (Μάιος 2021), pp. 1-52, Ascon v1.2: Submission to NIST, https://ascon.iaik.tugraz.at/
37. Beyne T., Chen Y. L., Dobraunig C., Mennink B., (Μάιος 2021), pp. 1-55, Elephant v2: KU Leuven and imec-COSIC, Belgium, KU Leuven and imec-COSIC, Belgium, Lamarr Security Research, Austria, Radboud University, The Netherlands
38. Banik S., Chakrabarti A., Iwata T., Kazuhiko M., l Nandi M., Peyrin T., Sasaki Y. S. , Siang M., Yosuke T., (Μάιος 2021), pp.1-36, GIFT-COFB v1.1
39. Hell M., Johansson T., Maximov A., Meier W., Yoshida H., pp. 1-15, Grain-128AEAD, Round 3 Tweak and Motivation
40. Hell M., Johansson T., Meier W., Sonnerup J., Yoshida H., (2019), pp. 55-71, An AEAD Variant of the Grain Stream Cipher, vol11445, Dept. of Electrical and Information Technology, Lund University, Sweden, FHNW, Windisch, Switzerland, Cyber Physical Security Research Center, National Institute of Advanced Industrial Science and Technology (AIST), Japan, http://link.springer.com/10.1007/978-3-030-16458-4_5, ISBN: 978-3-030-16457-7 978-3-030-16458-4
41. Dobraunig C., Eichlseder M., Mangard S., Mendel F., Mennink B., Primas R, Unterluggauer T., (Σεπτέμβριος 2020), pp. 1-14, NIST Update: ISAP v2.0, https://isap.iaik.tugraz.at/
42. Dobraunig C., Eichlseder M., Mangard S., Mendel F., Mennink B., Primas R, Unterluggauer T., (Μάιος 2021), pp. 1-54, ISAP v2.0, https://isap.iaik.tugraz.at/, https://isap.iaik.tugraz.at
43. ISAP Code Package: https://github.com/isap-lwc/isap-code-package
44. Bao Z., Chakraborti A., Datta N., Guo J., l Nandi M., Peyrin T., Yasuda K., (Μάιος 2021), pp.1-17, PHOTON-Beetle Authenticated Encryption and Hash Family
45. PHOTON-Beetle Authenticated Encryption: https://github.com/isap-lwc/isap-code-package
46. Guo C., Iwata T., Khairallah M., Minematsu K., Peyrin T. (Σεπτέμβριος 2022), pp.1-7, Final-round updates on Romulus, :https://romulusae.github.io/romulus/
47. Romulus Authenticated Encryption / Hash: https://romulusae.github.io/romulus/
48. Beirle C., Biryukov A., Cardoso dos Santos L., Großschadl J., Moradi A., Perrin L., Rezaei Shahmirzadi A., Udovenko A., Velichkov V., Wang Q, (Σεπτέμβριος 2022), pp.1-6, An Update on the LWC Finalist Sparkle
49. https://sparkle-lwc.github.io : A collection of lightweight symmetric cryptographic primitives, finalist of the ongoing NIST lightweight standardisation effort.
50. Wu H., Huang T. (Σεπτέμβριος 2022), pp. 1-7, TinyJAMBU Update, Division of Mathematical Sciences, Nanyang Technological University, https://csrc.nist.gov/csrc/media/Projects/lightweight-

cryptography/documents/finalist-round/status-updates/tinyjambu-update.pdf
51. Beyne T., Chen Y. L., Dobraunig C, Mennink B., (2020), pp.5-30, Dumbo, Jumbo, and Delirium: Parallel Authenticated Encryption for theLightweight Circus, Vol. 2020, No. S1, KU Leuven and imec-COSIC, Leuven, Belgium2Radboud University, Nijmegen, The Netherlands, fhttps://doi.org/10.13154/tosc.v2020.iS1.5-30, ISSN 2519-173X
52. Daemen J., Hoffert S., Mella S., Peeters M., Assche G. V., Ronny V. K., (Σεπτέμβριος 2022), pp.1-6, Xoodyak, a final update, Radboud University, STMicroelectronics, https://tosc.iacr.org/index.php/ToSC/article/view/8616/8182
53. Bertoni G., Daemen J., Hoffert S., Mella S., Peeters M., Assche G. V., Ronny V. K, Xoodyak: 1STMicroelectronics - 2Radboud University - 3Security Pattern, https://keccak.team/xoodyak.html
54. TLS Basics: https://www.internetsociety.org/deploy360/tls/basics/
55. The Most Important Security Problems with IoT Devices: https://www.internetsociety.org/deploy360/tls/basics/
56. . Nofal R. A, Tran N., Carlos G., Liu Y., Dezfouli B., A Comprehensive Empirical Analysis of TLS Handshake, (Νοέμβριος 2019), pp.1-10, https://www.cse.scu.edu/~bdezfouli/publication/MSWIM2019_SCU_SIOTLAB.pdf
57. Biryukov A., Perrin L., State of the Art in Lightweight Symmetric Cryptography, pp.1-55, SnT, CSC, University of Luxembourg, https://eprint.iacr.org/2017/511.pdf
58. McKay K. A., Bassham L., Turan M. S., Mouha N., Report on Lightweight Cryptography (Μάρτιος 2017), National Institute of Standards and Technology, https://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf, DOI: 10.6028/NIST.IR.8114
59. The Transport Layer Security (TLS) Protocol Version 1.3, (2018) https://datatracker.ietf.org/doc/html/rfc8446 , ISSN: 2070-1721
60. Sung S., (Νοέμβριος 2020), An Introduction to Cipher Suites, https://www.keyfactor.com/blog/cipher-suites-explained/
61. 'Taking a Closer Look at the SSL Handshake', Hashed Out by The SSL StoreTM, (Απρίλιος 2019). https://www.thesslstore.com/blog/explaining-ssl-handshake/
62. A complete overview of SSL/TLS and its cryptographic system, (Ιούνιος 2021) https://dev.to/techschoolguru/a-complete-overview-of-ssl-tls-and-its-cryptographic-system-36pd
63. Turan M. S., Kerry M., Chang D., Cagdas C. B. L., Kang J, Kelsey J., (Ιούλιος 2021), pp. 1-92, Status Report on the Second Round of the NIST Lightweight Cryptography Standardization Process, https://nvlpubs.nist.gov/nistpubs/ir/2021/NIST.IR.8369.pdf
64. Madushan H., Salam I., Alawatugoda J., Department of Computer Engineering, University of Peradeniya, Peradeniya 20400, Sri Lanka, School of Computing and Data Science, Xiamen University Malaysia,

Sepang 43900, Malaysia, Research & Innovation Centers, Faculty of Resilience, Rabdan Academy, Abu Dhabi P.O. Box 114646, United Arab Emirates, Institute for Integrated and Intelligent Systems, Griffith University, Nathan, QLD 4111, Australia : A Review of the NIST Lightweight Cryptography Finalists and Their Fault Analyses, (Δεκέμβριος 2022), 11(24), https://doi.org/10.3390/electronics11244199, https://www.mdpi.com/2079-9292/11/24/4199

65. R. Weatherley, https://github.com/rweather/lwc-finalists (accessed Μάιος 8/5/2023)
66. Manifavas C., Hatzivasilis G., Fysarakis K., Papaefstathiou Y., Department of Electrical Engineering and Computing Sciences, Rochester Institute of Technology Dubai, Techno Piont Building, Dubai Silicon Oasis, 341055, Dubai UAE, - Department of Electronic and Computer Engineering, Technical University of Crete, Akrotiri Campus, 73100 Chania, Crete Greece, (Δεκέμβριος 2015), A survey of lightweight stream ciphers for embedded systems. https://onlinelibrary.wiley.com/doi/epdf/10.1002/sec.1399
67. Hira R., Kitahara T., Miyahara D., Azumi H. Y., Li Y., Sakiyama K., The University of Electro-communications, Chofu, Tokyo 182-8585, Japan / Tokyo Institute of Technology, Fuchu, Tokyo 183-8538, Japan, (2021) Software Evaluation for Second Round Candidates in NIST Lightweight Cryptography., https://eprint.iacr.org/2022/591.pdf
68. BSI – Technische Richtlinie, Kryptographische Verfahren: Empfehlungen und Schlüssellängen, (Ιανουάριος 2023), Bundesamt für Sicherheit in der Informationstechnik, Postfach 20 03 63, 53133 Bonn, Germany, https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR
69. Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process, https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf
70. Buchanan, William J (2023). Light-weight Encryption and Hashing Performance Tests. Asecuritysite.com. https://asecuritysite.com/light/light
71. EL-hajj M., Mousawi H., Fadlallah A., (Ιανουάριος 2023), Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands, Faculty of Engineering, Lebanese University, Beirut 1533, Lebanon, Faculty of Arts and Science, University of Sciences and Arts in Lebanon, Beirut 1002, Lebanon: Analysis of Lightweight Cryptographic Algorithms on IoT Hardware Platform, 15(2), 54; https://doi.org/10.3390/fi15020054, https://www.mdpi.com/1999-5903/15/2/54#FD6-futureinternet-15-00054
72. Mohajerani K., Beckwith L., Abdulgadir A., Ferrufino E., Kaps J. P. and Gaj K., 1Cryptographic Engineering Research Group, George Mason University

Fairfax, VA, U.S.A, SCA Evaluation and Benchmarking of Finalists in the NIST Lightweight Cryptography Standardization Process, https://eprint.iacr.org/2023/484.pdf
73. https://www.arduino.cc/en/software
74. Nist,https://csrc.nist.gov/Projects/lightweight-cryptography/finalists
75. https://www.allkeysgenerator.com/Random/Security-Encryption-Key-Generator.aspx

ΠΑΡΑΡΤΗΜΑ Α
Ακρωνύμια

- AD - Associated Data
- AEAD - Authenticated Encryption With Additional Data
- C - Cyphertext
- DH - Diffie-Hellman
- DHE - Diffie-Hellman Ephemeral
- EDH -Elliptic-Diffie–Hellman
- GE – Gate Equivalents
- HEX - Heximal
- HTTP - Hypertext Transfer Protocol
- HTTPS - Hypertext Transfer Protocol Secure
- IANA – Internet Assigned Numbers Authority
- IETF - Internet Engineering Task Force
- IDE - Integrated Development Environment
- IOT – Internet of Things
- IV – Initialization Vector
- LWC – Lightweight Cryptography
- M – Message
- MAC – Message Authentication Code
- NIST - National Institute of Standards and Technology
- RSA - Rivest–Shamir–Adleman
- SMTP - Simple Mail Transfer Protocol
- SPN – Substitution Permutation Network
- SSL – Secure Socket Layer
- TLS – Transport Layer Security

ΠΑΡΑΡΤΗΜΑ Β
Κώδικας

Τα αρχεία κώδικα υπάρχουν στο αποθετήριο https://github.com/rweather/ascon-suite, https://github.com/rweather/lwc-finalists της πλατφόρμας Git Hub, στο επίσημο σάιτ https://csrc.nist.gov/Projects/lightweight-cryptography/finalists και ο κώδικας του αρχείου .ino (sketch ASCON) δημιουργήθηκε τοπικά μέσα στο φάκελο libraries του Arduino ώστε να τρέξει σε περιβάλλον συσκευής χαμηλών πόρων για δοκιμές όπου ακολουθεί παρακάτω, Επισημαίνεται ότι πρέπει να γίνει λήψη και του επίσημου κώδικα των αλγορίθμων.

- Αρχείο ASCON.ino

```
#include <ASCON.h>
#define crypto_feed() do { ; } while (0)

typedef void (*aead_cipher_encrypt_t)
   (unsigned char *c, size_t *clen,
    const unsigned char *m, size_t mlen,
    const unsigned char *ad, size_t adlen,
    const unsigned char *npub,
    const unsigned char *k);
typedef int (*aead_cipher_decrypt_t)
   (unsigned char *m, size_t *mlen,
    const unsigned char *c, size_t clen,
    const unsigned char *ad, size_t adlen,
    const unsigned char *npub,
    const unsigned char *k);
typedef void (*aead_cipher_pk_init_t)
   (unsigned char *pk, const unsigned char *k);
typedef void (*aead_cipher_pk_free_t)(unsigned char *pk);
typedef void (*aead_hash_t)
   (unsigned char *out, const unsigned char *in, size_t inlen);

#define DEFAULT_PERF_LOOPS 1000
#define DEFAULT_PERF_LOOPS_16 3000
#define DEFAULT_PERF_HASH_LOOPS 1000

static int PERF_LOOPS = DEFAULT_PERF_LOOPS;
static int PERF_LOOPS_16 = DEFAULT_PERF_LOOPS_16;
static int PERF_HASH_LOOPS = DEFAULT_PERF_HASH_LOOPS;
static bool PERF_MASKING = false;

#define MAX_DATA_SIZE 128
#define MAX_TAG_SIZE 16

static unsigned char const key[16] = {
   0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C,
0x1D, 0x1E, 0x1F
};
static unsigned char const nonce[16] = {
   0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C,
0x0D, 0x0E, 0x0F
};
static unsigned char plaintext[MAX_DATA_SIZE];
static unsigned char ciphertext[MAX_DATA_SIZE + MAX_TAG_SIZE];

void perfCipherEncrypt128
   (aead_cipher_encrypt_t encrypt,
    unsigned key_size,
    aead_cipher_pk_init_t init_key,
    aead_cipher_pk_free_t free_key)
```

```
{
   const unsigned char *k = key;
   unsigned char pk[key_size + 8];
   unsigned long start;
   unsigned long elapsed;
   size_t len;
   int count;

   for (count = 0; count < MAX_DATA_SIZE; ++count)
      plaintext[count] = (unsigned char)count;

   Serial.print("   encrypt 128 byte packets ... ");

   if (init_key) {
      init_key(pk, key);
      k = pk;
   }

   start = micros();
   for (count = 0; count < PERF_LOOPS; ++count) {
      encrypt(ciphertext, &len, plaintext, 128, 0, 0, nonce, k);
   }
   elapsed = micros() - start;

   if (free_key)
      free_key(pk);

   Serial.print(elapsed / (128.0 * PERF_LOOPS));
   Serial.print("us per byte, ");
   Serial.print((128.0 * PERF_LOOPS * 1000000.0) / elapsed);
   Serial.println(" bytes per second");
}

void perfCipherDecrypt128
   (aead_cipher_encrypt_t encrypt,
    aead_cipher_decrypt_t decrypt,
    unsigned key_size,
    aead_cipher_pk_init_t init_key,
    aead_cipher_pk_free_t free_key)
{
   const unsigned char *k = key;
   unsigned char pk[key_size + 8];
   unsigned long start;
   unsigned long elapsed;
   size_t clen;
   size_t plen;
   int count;

   for (count = 0; count < MAX_DATA_SIZE; ++count)
```

```
    plaintext[count] = (unsigned char)count;
  encrypt(ciphertext, &clen, plaintext, 128, 0, 0, nonce, k);

  Serial.print("   decrypt 128 byte packets ... ");

  if (init_key) {
    init_key(pk, key);
    k = pk;
  }

  start = micros();
  for (count = 0; count < PERF_LOOPS; ++count) {
    decrypt(plaintext, &plen, ciphertext, clen, 0, 0, nonce, k);
  }
  elapsed = micros() - start;

  if (free_key)
    free_key(pk);

  Serial.print(elapsed / (128.0 * PERF_LOOPS));
  Serial.print("us per byte, ");
  Serial.print((128.0 * PERF_LOOPS * 1000000.0) / elapsed);
  Serial.println(" bytes per second");
}

void perfCipherEncrypt16
  (aead_cipher_encrypt_t encrypt,
   unsigned key_size,
   aead_cipher_pk_init_t init_key,
   aead_cipher_pk_free_t free_key)
{
  const unsigned char *k = key;
  unsigned char pk[key_size + 8];
  unsigned long start;
  unsigned long elapsed;
  size_t len;
  int count;

  for (count = 0; count < MAX_DATA_SIZE; ++count)
    plaintext[count] = (unsigned char)count;

  Serial.print("   encrypt  16 byte packets ... ");

  if (init_key) {
    init_key(pk, key);
    k = pk;
  }

  start = micros();
```

```
    for (count = 0; count < PERF_LOOPS_16; ++count) {
        encrypt(ciphertext, &len, plaintext, 16, 0, 0, nonce, k);
    }
    elapsed = micros() - start;

    if (free_key)
        free_key(pk);

    Serial.print(elapsed / (16.0 * PERF_LOOPS_16));
    Serial.print("us per byte, ");
    Serial.print((16.0 * PERF_LOOPS_16 * 1000000.0) / elapsed);
    Serial.println(" bps");
}

void perfCipherDecrypt16
    (aead_cipher_encrypt_t encrypt,
     aead_cipher_decrypt_t decrypt,
     unsigned key_size,
     aead_cipher_pk_init_t init_key,
     aead_cipher_pk_free_t free_key)
{
    const unsigned char *k = key;
    unsigned char pk[key_size + 8];
    unsigned long start;
    unsigned long elapsed;
    size_t clen;
    size_t plen;
    int count;

    for (count = 0; count < MAX_DATA_SIZE; ++count)
        plaintext[count] = (unsigned char)count;
    encrypt(ciphertext, &clen, plaintext, 16, 0, 0, nonce, k);

    Serial.print("   decrypt  16 byte packets ... ");

    if (init_key) {
        init_key(pk, key);
        k = pk;
    }

    start = micros();
    for (count = 0; count < PERF_LOOPS_16; ++count) {
        decrypt(plaintext, &plen, ciphertext, clen, 0, 0, nonce, k);
    }
    elapsed = micros() - start;

    if (free_key)
        free_key(pk);
```

```
    Serial.print(elapsed / (16.0 * PERF_LOOPS_16));
    Serial.print("us per byte, ");
    Serial.print((16.0 * PERF_LOOPS_16 * 1000000.0) / elapsed);
    Serial.println(" bytes per second");
}

bool equal_hex(const char *expected, const unsigned char *actual, unsigned len)
{
    int ch, value;
    while (len > 0) {
        if (expected[0] == '\0' || expected[1] == '\0')
            return false;
        ch = *expected++;
        if (ch >= '0' && ch <= '9')
            value = (ch - '0') * 16;
        else if (ch >= 'A' && ch <= 'F')
            value = (ch - 'A' + 10) * 16;
        else if (ch >= 'a' && ch <= 'f')
            value = (ch - 'a' + 10) * 16;
        else
            return false;
        ch = *expected++;
        if (ch >= '0' && ch <= '9')
            value += (ch - '0');
        else if (ch >= 'A' && ch <= 'F')
            value += (ch - 'A' + 10);
        else if (ch >= 'a' && ch <= 'f')
            value += (ch - 'a' + 10);
        else
            return false;
        if (actual[0] != value)
            return false;
        ++actual;
        --len;
    }
    return len == 0;
}


void perfCipher(const char *name, aead_cipher_encrypt_t encrypt,
            aead_cipher_decrypt_t decrypt, const char *sanity_vec)
{
    crypto_feed();
    Serial.print(name);
    Serial.print(':');
    Serial.println();
    perfCipherEncrypt128(encrypt, 0, 0, 0);
    perfCipherDecrypt128(encrypt, decrypt, 0, 0, 0);
    perfCipherEncrypt16(encrypt, 0, 0, 0);
```

```
    perfCipherDecrypt16(encrypt, decrypt, 0, 0, 0);

    Serial.println();
}

void perfCipherPK(const char *name, unsigned key_size,
                  aead_cipher_encrypt_t encrypt,
                  aead_cipher_decrypt_t decrypt,
                  aead_cipher_pk_init_t init_key,
                  aead_cipher_pk_free_t free_key,
                  const char *sanity_vec)
{
    crypto_feed();
    Serial.print(name);
    Serial.print(':');
    Serial.println();
    perfCipherEncrypt128(encrypt, key_size, init_key, free_key);
    perfCipherDecrypt128(encrypt, decrypt, key_size, init_key, free_key);
    perfCipherEncrypt16(encrypt, key_size, init_key, free_key);
    perfCipherDecrypt16(encrypt, decrypt, key_size, init_key, free_key);

    Serial.println();
}
#define HASH_BUFSIZ 512

static unsigned char hash_buffer[HASH_BUFSIZ];

void perfHash_N(aead_hash_t hash_func, int size)
{
    unsigned long start;
    unsigned long elapsed;
    unsigned long long len;
    int count, loops;

    for (count = 0; count < size; ++count)
        hash_buffer[count] = (unsigned char)count;

    Serial.print("   hash ");
    if (size < 1000) {
        if (size < 100)
            Serial.print("  ");
        else
            Serial.print(" ");
    }
    Serial.print(size);
    Serial.print(" bytes");

    if (size < HASH_BUFSIZ)
        loops = PERF_HASH_LOOPS * 4;
```

```
    else
        loops = PERF_HASH_LOOPS;

    start = micros();
    for (count = 0; count < loops; ++count) {
        hash_func(ciphertext, hash_buffer, size);
    }
    elapsed = micros() - start;

    Serial.print(elapsed / (((double)size) * loops));
    Serial.print("us per byte, ");
    Serial.print((1000000.0 * size * loops) / elapsed);
    Serial.println(" bps");
}

void perfHash(const char *name, aead_hash_t hash_func)
{
    crypto_feed();
    Serial.print(name);
    Serial.print(':');
    Serial.println();

    perfHash_N(hash_func, HASH_BUFSIZ);
    perfHash_N(hash_func, 128);
    perfHash_N(hash_func, 16);

    Serial.println();
}

void setup()
{

    Serial.begin(9600);
    Serial.println();(size = 11)
    perfCipher("ASCON-128", ascon128_aead_encrypt, ascon128_aead_decrypt,

"76807B6448896CE58842CB4AED6C41041D6DEC3B3A0DD69901F988A337A
7239C411A18313622FC");
    perfCipher("ASCON-128a", ascon128a_aead_encrypt, ascon128a_aead_decrypt,

"C52E4E39F5EF9F8461912AED7ABBA1B8EB8AD7ACD54637D193C53712797
53F2177BFC76E5FC300")
    perfHash("ASCON-HASH", ascon_hash);
    perfHash("ASCON-HASHA", ascon_hasha);
    perfHash("ASCON-XOF", ascon_xof);
    perfHash("ASCON-XOFA", ascon_xofa);

    perfCipherPK("ASCON-128-masked", sizeof(ascon_masked_key_128_t),
            (aead_cipher_encrypt_t)ascon128_masked_aead_encrypt,
```

```
            (aead_cipher_decrypt_t)ascon128_masked_aead_decrypt,
            (aead_cipher_pk_init_t)ascon_masked_key_128_init,
            (aead_cipher_pk_free_t)ascon_masked_key_128_free,

"76807B6448896CE58842CB4AED6C41041D6DEC3B3A0DD69901F988A337A
7239C411A18313622FC");
   perfCipherPK("ASCON-128a-masked", sizeof(ascon_masked_key_128_t),
            (aead_cipher_encrypt_t)ascon128a_masked_aead_encrypt,
            (aead_cipher_decrypt_t)ascon128a_masked_aead_decrypt,
            (aead_cipher_pk_init_t)ascon_masked_key_128_init,
            (aead_cipher_pk_free_t)ascon_masked_key_128_free,

"C52E4E39F5EF9F8461912AED7ABBA1B8EB8AD7ACD54637D193C53712797
53F2177BFC76E5FC300");
 }

void loop()
{
}
```

- Αρχείο .ino ανάλογα τον αλγόριθμο που θα τρέξει συμπληρώνεται αντίστοιχα το κλειδί και το nonce και θα πρέπει να μπουν σε σχόλια τα υπόλοιπα

```
#include "aead-metadata.h"
#include "internal-sha256.h"
#include "internal-masking.h"

#define crypto_feed() do { ; } while (0)

#define DEFAULT_PERF_LOOPS 1000
#define DEFAULT_PERF_LOOPS_16 3000
#define DEFAULT_PERF_HASH_LOOPS 1000

static int PERF_LOOPS = DEFAULT_PERF_LOOPS;
static int PERF_LOOPS_16 = DEFAULT_PERF_LOOPS_16;
static int PERF_HASH_LOOPS = DEFAULT_PERF_HASH_LOOPS;
static bool PERF_MASKING = false;

#define MAX_DATA_SIZE 128
#define MAX_TAG_SIZE 32 /*Συμπληρώνεται το μέγεθος της ετικέτας - αρχείο
όνομαΑλγόριθμου.h ανάλογα τον κώδικα (library αποθετήριο φάκελος src)*/

static unsigned char const key[16] = {
  //Συμπληρώνεται το αντίστοιχο κλειδί

};
static unsigned char const nonce[16] = {
//Συμπληρώνεται το αντίστοιχο nonce
```

```
};
static unsigned char plaintext[MAX_DATA_SIZE];
static unsigned char ciphertext[MAX_DATA_SIZE + MAX_TAG_SIZE];

static unsigned long encrypt_128_time = 0;
static unsigned long encrypt_16_time = 0;
static unsigned long decrypt_128_time = 0;
static unsigned long decrypt_16_time = 0;
static unsigned long encrypt_128_ref = 0;
static unsigned long encrypt_16_ref = 0;
static unsigned long decrypt_128_ref = 0;
static unsigned long decrypt_16_ref = 0;
static unsigned long hash_1024_time = 0;
static unsigned long hash_128_time = 0;
static unsigned long hash_16_time = 0;
static unsigned long hash_1024_ref = 0;
static unsigned long hash_128_ref = 0;
static unsigned long hash_16_ref = 0;

static void print_x(double value)
{
  if (value < 0.005)
    Serial.print(value, 4);
  else
    Serial.print(value);
}

void perfCipherEncrypt128(const aead_cipher_t *cipher)
{
  unsigned long start;
  unsigned long elapsed;
  size_t len;
  int count;

  for (count = 0; count < MAX_DATA_SIZE; ++count)
    plaintext[count] = (unsigned char)count;

  Serial.print("   encrypt 128 byte packets");

  start = micros();
  for (count = 0; count < PERF_LOOPS; ++count) {
    cipher->encrypt
      (ciphertext, &len, plaintext, 128, 0, 0, nonce, key);
  }
  elapsed = micros() - start;
  encrypt_128_time = elapsed;

  if (encrypt_128_ref != 0 && elapsed != 0) {
    print_x(((double)encrypt_128_ref) / elapsed);
```

```
        Serial.print("x, ");
    }

    Serial.print(elapsed / (128.0 * PERF_LOOPS));
    Serial.print(" use per byte  DYO, ");
    Serial.print((128.0 * PERF_LOOPS * 1000000.0) / elapsed);
    Serial.println(" bytes per second");
}

void perfCipherDecrypt128(const aead_cipher_t *cipher)
{
    unsigned long start;
    unsigned long elapsed;
    size_t clen;
    size_t plen;
    int count;

    for (count = 0; count < MAX_DATA_SIZE; ++count)
        plaintext[count] = (unsigned char)count;
    cipher->encrypt(ciphertext, &clen, plaintext, 128, 0, 0, nonce, key);

    Serial.print("   decrypt 128 byte packets");

    start = micros();
    for (count = 0; count < PERF_LOOPS; ++count) {
        cipher->decrypt
            (plaintext, &plen, ciphertext, clen, 0, 0, nonce, key);
    }
    elapsed = micros() - start;
    decrypt_128_time = elapsed;

    if (decrypt_128_ref != 0 && elapsed != 0) {
        print_x(((double)decrypt_128_ref) / elapsed);
        Serial.print("x, ");
    }

    Serial.print(elapsed / (128.0 * PERF_LOOPS));
    Serial.print(" us per byte, ");
    Serial.print((128.0 * PERF_LOOPS * 1000000.0) / elapsed);
    Serial.println(" bps");
}


void perfCipherEncrypt16(const aead_cipher_t *cipher)
{
    unsigned long start;
    unsigned long elapsed;
    size_t len;
    int count;
```

```
    for (count = 0; count < MAX_DATA_SIZE; ++count)
        plaintext[count] = (unsigned char)count;

    Serial.print("   encrypt  16 byte");

    start = micros();
    for (count = 0; count < PERF_LOOPS_16; ++count) {
        cipher->encrypt
            (ciphertext, &len, plaintext, 16, 0, 0, nonce, key);
    }
    elapsed = micros() - start;
    encrypt_16_time = elapsed;

    if (encrypt_16_ref != 0 && elapsed != 0) {
        print_x(((double)encrypt_16_ref) / elapsed);
        Serial.print("x, ");
    }

    Serial.print(elapsed / (16.0 * PERF_LOOPS_16));
    Serial.print(" us per byte,EFTA ");
    Serial.print((16.0 * PERF_LOOPS_16 * 1000000.0) / elapsed);
    Serial.println(" bps");
}

void perfCipherDecrypt16(const aead_cipher_t *cipher)
{
    unsigned long start;
    unsigned long elapsed;
    size_t clen;
    size_t plen;
    int count;

    for (count = 0; count < MAX_DATA_SIZE; ++count)
        plaintext[count] = (unsigned char)count;
    cipher->encrypt(ciphertext, &clen, plaintext, 16, 0, 0, nonce, key);

    Serial.print("   decrypt  16 byte packets ENNIA... ");

    start = micros();
    for (count = 0; count < PERF_LOOPS_16; ++count) {
        cipher->decrypt
            (plaintext, &plen, ciphertext, clen, 0, 0, nonce, key);
    }
    elapsed = micros() - start;
    decrypt_16_time = elapsed;

    if (decrypt_16_ref != 0 && elapsed != 0) {
        print_x(((double)decrypt_16_ref) / elapsed);
```

```
      Serial.print("x, ");
    }

    Serial.print(elapsed / (16.0 * PERF_LOOPS_16));
    Serial.print(" us per byte");
    Serial.print((16.0 * PERF_LOOPS_16 * 1000000.0) / elapsed);
    Serial.println(" bps");
}

bool equal_hex(const char *expected, const unsigned char *actual, unsigned len)
{
    int ch, value;
    while (len > 0) {
        if (expected[0] == '\0' || expected[1] == '\0')
            return false;
        ch = *expected++;
        if (ch >= '0' && ch <= '9')
            value = (ch - '0') * 16;
        else if (ch >= 'A' && ch <= 'F')
            value = (ch - 'A' + 10) * 16;
        else if (ch >= 'a' && ch <= 'f')
            value = (ch - 'a' + 10) * 16;
        else
            return false;
        ch = *expected++;
        if (ch >= '0' && ch <= '9')
            value += (ch - '0');
        else if (ch >= 'A' && ch <= 'F')
            value += (ch - 'A' + 10);
        else if (ch >= 'a' && ch <= 'f')
            value += (ch - 'a' + 10);
        else
            return false;
        if (actual[0] != value)
            return false;
        ++actual;
        --len;
    }
    return len == 0;
}
void perfCipher(const aead_cipher_t *cipher, const char *sanity_vec)
{
    crypto_feed_watchdog();
    Serial.print(cipher->name);
    Serial.print(':');
    Serial.println();

    if (sanity_vec)
//      perfCipherSanityCheck(cipher, sanity_vec);
```

```
    perfCipherEncrypt128(cipher);
    perfCipherDecrypt128(cipher);
    perfCipherEncrypt16(cipher);
    perfCipherDecrypt16(cipher);

    if (encrypt_128_ref != 0) {
        unsigned long ref_avg = encrypt_128_ref + decrypt_128_ref +
                                encrypt_16_ref  + decrypt_16_ref;
        unsigned long time_avg = encrypt_128_time + decrypt_128_time +
                                 encrypt_16_time  + decrypt_16_time;
        Serial.print("   average ... ");
        print_x(((double)ref_avg) / time_avg);
        Serial.print("x");
        if (PERF_MASKING) {
            Serial.print(" = 1 / ");
            print_x(((double)time_avg) / ref_avg);
            Serial.print("x");
        }
        Serial.println();
    }

    Serial.println();
}

static unsigned char hash_buffer[1024];

unsigned long perfHash_N
    (const aead_hash_algorithm_t *hash_alg, int size, unsigned long ref)
{
    unsigned long start;
    unsigned long elapsed;
    int count, loops;

    for (count = 0; count < size; ++count)
        hash_buffer[count] = (unsigned char)count;

    Serial.print("   hash ");
    if (size < 1000) {
        if (size < 100)
            Serial.print("  ");
        else
            Serial.print(" ");
    }
    Serial.print(size);
    Serial.print(" bytes ");

        if (size < 1024)
        loops = PERF_HASH_LOOPS * 4;
```

```
    else
        loops = PERF_HASH_LOOPS;

    start = micros();
    for (count = 0; count < loops; ++count) {
        hash_alg->hash(ciphertext, hash_buffer, size);
    }
    elapsed = micros() - start;

    if (ref != 0 && elapsed != 0) {
        print_x(((double)ref) / elapsed);
        Serial.print("x, ");
    }

    Serial.print(elapsed / (((double)size) * loops));
    Serial.print(" us per byte"); //dekatria
    Serial.print((1000000.0 * size * loops) / elapsed);
    Serial.println(" bps");//bytes per second

    return elapsed;
}

void perfHash(const aead_hash_algorithm_t *hash_alg, const char *sanity_vec)
{
    crypto_feed_watchdog();
    Serial.print(hash_alg->name);
    Serial.print(':');
    Serial.println();

    if (sanity_vec)

    hash_1024_time = perfHash_N(hash_alg, 1024, hash_1024_ref);
    hash_128_time = perfHash_N(hash_alg, 128, hash_128_ref);
    hash_16_time = perfHash_N(hash_alg, 16, hash_16_ref);

    if (hash_16_ref != 0) {
        double avg = ((double)hash_1024_ref) / hash_1024_time;
        avg += ((double)hash_128_ref) / hash_128_time;
        avg += ((double)hash_16_ref) / hash_16_time;
        avg /= 3.0;
        Serial.print("   average ");
        print_x(avg);
        Serial.print("x");
        Serial.println();
    }

    Serial.println();
}
```

```
void perfMasked(const aead_cipher_t *ref_cipher,
            const aead_cipher_t *masked_cipher)
{
   encrypt_128_ref = 0;
   decrypt_128_ref = 0;
   encrypt_16_ref = 0;
   decrypt_16_ref = 0;
   perfCipher(ref_cipher, 0);
   encrypt_128_ref = encrypt_128_time;
   decrypt_128_ref = decrypt_128_time;
   encrypt_16_ref = encrypt_16_time;
   decrypt_16_ref = decrypt_16_time;
   Serial.print("[");
   Serial.print(AEAD_MASKING_SHARES);
   Serial.print("] ");
   perfCipher(masked_cipher, 0);
}

void setup()
{
   Serial.begin(9600);
   Serial.println();


   perfCipher(&delirium_cipher,
"1EBBE29D3EC4D574840905EFCEBFB40D02E1AB1B8B9994B8E19B5C7E461
C77D276842CF6BEE6EA");
  perfCipher(&gift_cofb_cipher,
"ABC3924173986D9EAA16CE0D01E923E5B6B26DC70E2190FB0E95FF754FF1
A6943770CA3C04958A");
   perfCipher(&grain128_aead_cipher,
"A4AB16F5B985B23EE9839C86A573B149D64EA150FEC21A81FD32406809D
D51");

  perfCipher(&photon_beetle_128_cipher,
"687B6BFD3807B447E418C8006C87A375AD55CEC555FA154A73EE361B62BB
DA16875EDE631F445D");
   perfCipher(&photon_beetle_32_cipher,
"05780949CD88CDC5940C408DD9ED28DD912386D437484DE5D4F65D10397
CCE9E19F203840ACF2D");

  perfCipher(&romulus_n_cipher,
"B0C179AC69E8583FD66B5C00368D4DBD93157CF52B93769A1EC2DF4019D
E6D26A2FF2D31063F28");
   perfCipher(&romulus_m_cipher,
"C21701C35E0E5FB450C66BD785B5E8A35426198531AD9BF1B30BB9ACC229
A49C7C247BD28887DC");
```

```
    perfCipher(&romulus_t_cipher,
"14431457C1B573058A16B8A10880FE96EF6ACAE8259E14523291D603D3A0
066229A670554E094C");
   perfCipher(&schwaemm_256_128_cipher,
"FA127C39BB1AB15429F59EF32F2742DB80A7F7A26939101E42502D7FB826
73CF4977F6C6E12658");
   perfCipher(&schwaemm_192_192_cipher,
"AED467CB67699D64AB5CE6AC4D578AA6C11AA962F639491095FD7DA7C3F
E384B748518E9EEF24A4FF088466D3BE83B");
   perfCipher(&schwaemm_128_128_cipher,
"8FC6A5B02165D2B9FF5838B24C7CFFC89F1A4BCB0AE9D1BEBBDAF0E435E
F3D3B1E88283A992ADC");
   perfCipher(&schwaemm_256_256_cipher,
"208CC82C35AF6227C7CF5C96A71BFBF10227D457DBD613F816C7704BA4AF
F2E520BB179DAA1883D94212C18FD70EDDA2341E6058738F28");

  perfCipher(&tiny_jambu_128_cipher,
"E30F24BBFC434EB18B92A3A4742BBAE61383F62BC9104E976569195FE559
BC");
perfCipher(&tiny_jambu_192_cipher,
"317B8563AFA9B731FDF1F29FA688D0B0280422844CFEBAEE75CCE206898F
65");
 perfCipher(&tiny_jambu_256_cipher,
"D38B7389554B9C5DD8CA961C42CBE0017B102D0E01B82E91EAB122742F5
8F9");

   perfCipher(&xoodyak_cipher,
"0E193FA578653462B128754C9CE9E5E4BB0910CA40C91A247E4EDCF2EC35
E9098AF34EDF147366");

    // Performance of masked ciphers on their own.
 perfCipher(&gift_cofb_masked_cipher,
"ABC3924173986D9EAA16CE0D01E923E5B6B26DC70E2190FB0E95FF754FF1
A6943770CA3C04958A");

  perfCipher(&tiny_jambu_128_masked_cipher,
"E30F24BBFC434EB18B92A3A4742BBAE61383F62BC9104E976569195FE559
BC");
  perfCipher(&tiny_jambu_192_masked_cipher,
"317B8563AFA9B731FDF1F29FA688D0B0280422844CFEBAEE75CCE206898F
65");
   perfCipher(&tiny_jambu_256_masked_cipher,
"D38B7389554B9C5DD8CA961C42CBE0017B102D0E01B82E91EAB122742F5
8F9");
   perfCipher(&xoodyak_masked_cipher,
"0E193FA578653462B128754C9CE9E5E4BB0910CA40C91A247E4EDCF2EC35
E9098AF34EDF147366");
```

```
  perfHash(&esch_256_hash_algorithm,
"E1F292177A096547DFDE7F1E2E33EFB6A7C4C6DAAA6AFC95C9521E5D13168AC3");
  perfHash(&esch_384_hash_algorithm,
"DCAD7D7394C64CB59BE79EE06A42FE5A420C5718156C6D3CC44ED07E699DDBE79BB2919D65EC4A24B5ECE4AFB11DFF54");
 perfHash(&photon_beetle_hash_algorithm,
"9DB4465229E011100FFA49C0500C3A7B2B154F29AFFD0291CA3EFF69A74DBA9E");
perfHash(&romulus_hash_algorithm,
"40055D86525079F0DB65F9DA46C6282D63B571C1DEE72BB3B5FB2C7319AB30EC");
  perfHash(&xoodyak_hash_algorithm,
"511AD3AA185ACC22EB141A81C1EBDA05EADA4E0C07BFBAD3A4855DB3E96C2164");
   encrypt_128_ref /= 10;
   decrypt_128_ref /= 10;
   encrypt_16_ref /= 10;
   decrypt_16_ref /= 10;
   PERF_LOOPS = DEFAULT_PERF_LOOPS / 10;
   PERF_LOOPS_16 = DEFAULT_PERF_LOOPS_16 / 10;
perfCipher(&dumbo_cipher,
"0867290AD29D219C4BF3BF0BD652099B499B5B9CD7401BB862073E167E6543");
perfCipher(&jumbo_cipher,
"AE5D4F2BFAE6D432A1B6E92EB8955A7F2FD61692B269CDB16F7CA74F04CFE1");
 perfCipher(&isap_ascon_128a_cipher,
"2CDE28DBBBD9131EBC568D77725B25937CF8EDB8A8F50A51312527CC6AEA52AED910035253C093");
 perfCipher(&isap_ascon_128_cipher,
"B8529BCE1B3F9D0DB7A9C8DD43DD35D18E41801A814A2946E3500BD4A77E3EFF16EFABD6CCA575");

   perfCipher(&isap_keccak_128a_cipher,
"01BC9CCB186E4A3732E86B9FAC4ABF3E6C4A8274A185FF1F7A1B9A98C623F126568CBADA74FAB5");
   perfCipher(&isap_keccak_128_cipher,
"59D5A45BCBCB332311869B73F633D29606056B791F8A68F20CA7C894D7CDE7A06B357814696787");

   PERF_LOOPS = DEFAULT_PERF_LOOPS / 10;
   PERF_LOOPS_16 = DEFAULT_PERF_LOOPS_16 / 10;
   PERF_MASKING = true;
perfMasked(&gift_cofb_cipher, &gift_cofb_masked_cipher);
 perfMasked(&tiny_jambu_128_cipher, &tiny_jambu_128_masked_cipher);
  perfMasked(&tiny_jambu_192_cipher, &tiny_jambu_192_masked_cipher);
   perfMasked(&tiny_jambu_256_cipher, &tiny_jambu_256_masked_cipher);
   perfMasked(&xoodyak_cipher, &xoodyak_masked_cipher);
```

```
}

void loop()
{
}
```