ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

" Εμπιστευτικότητα δεδομένων σε πραγματικό χρόνο με χρήση FPGA για εφαρμογές IoT "

ΝΙΚΟΛΑΟΣ (ΔΗΜΗΤΡΙΟΣ) ΜΑΥΡΟΓΙΩΡΓΗΣ

Α.Μ. 2874

ΕΠΙΒΛΕΠΩΝ: ΠΑΡΑΣΚΕΥΑΣ ΚΙΤΣΟΣ,ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ

ΠΑΤΡΑ 2023

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή
Πάτρα, Ημερομηνία

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1.

2.

3.

Υπεύθυνη Δήλωση Φοιτητή

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τη συγκεκριμένη εργασία.

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Πελοποννήσου δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

ΠΕΡΙΛΗΨΗ

Το Internet of Things (IoT) είναι μια τεχνολογία που αναφέρεται στη συνδεσιμότητα των συσκευών στο διαδίκτυο, ώστε να μπορούν να ανταλλάσσουν δεδομένα και να εκτελούν λειτουργίες χωρίς την ανάγκη της ανθρώπινης παρέμβασης. Η συνδεσιμότητα αυτών των συσκευών στο διαδίκτυο απαιτεί την εφαρμογή ασφάλειας και προστασίας των δεδομένων τους, κάτι το οποίο επιτυγχάνεται με τη χρήση αλγορίθμων κρυπτογραφίας όπως ο Grain-128AEADv2 καθώς και των Field Programmable Gate Arrays (FPGAs) ενός τύπου ολοκληρωμένου κυκλώματος για ταχύτερη επεξεργασία των δεδομένων. Ο Grain-128AEADv2 είναι ένας lightweight stream cipher, ο οποίος σχεδιάστηκε για να παρέχει πιστοποιημένη κρυπτογράφηση σε εφαρμογές IoT και βασίζεται στον Grain-128. Αναπτύχθηκε από τους Martin Hell, Thomas Johansson, Alexander Maximov, Willi Meier, και Hirotaka Yoshida. Σκοπός της παρούσας πτυχιακής εργασίας είναι η μελέτη του Grain-128AEADv2 και η ανάλυση του τρόπου λειτουργίας του. Όλα τα κομμάτια κώδικα που χρησιμοποιήθηκαν, επιβεβαιώθηκαν σε επίπεδο προσομοίωσης με το πρόγραμμα VIVADO 2021.2.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Internet of Things (IoT), Grain-128AEADv2, Field Programmable Gate Arrays (FPGAs), lightweight stream cipher

ABSTRACT

Internet of Things (IoT) is a technology that refers to the connectivity of devices to the internet, so that they can exchange data and perform tasks without the need for human intervention. The connectivity of these devices to the internet requires the implementation of security and protection of their data, which is achieved through the use of encryption algorithms such as Grain-128AEADv2 and Field Programmable Gate Arrays (FPGAs), a type of integrated circuits for faster data processing. Grain-128AEADv2 is a lightweight stream cipher that was designed to provide certified encryption in IoT applications and is based on Grain-128. It was developed by Martin Hell, Thomas Johansson, Alexander Maximov, Willi Meier, and Hirotaka Yoshida. The purpose of this thesis is to study Grain-128AEADv2 and analyze the way it works. All the code pieces used were verified at simulation level with the VIVADO 2021.2 program..

KEYWORDS: Internet of Things (IoT), Grain-128AEADv2, Field Programmable Gate Arrays (FPGAs), lightweight stream cipher

ΕΥΧΑΡΙΣΤΙΕΣ

Για τη διεκπεραίωση της παρούσας πτυχιακής εργασίας, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου Παρασκευά Κίτσο για τη συνεργασία μας και που με την πολύτιμη βοήθειά του συνέβαλε στην αρτιότερη εκπόνησή της.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

1.1. Εισαγωγή

Στο παρόν κεφάλαιο αναλύεται το αντικείμενο της πτυχιακής εργασίας, το οποίο ασχολείται με τη μελέτη του αλγορίθμου Grain-128AEADv2, ο οποίος είχε υποβληθεί στο Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας (NIST, National Institute of Standards and Technology), και είναι υλοποιημένος στη γλώσσα VHDL. Ο εν λόγω αλγόριθμος ανήκει στην οικογένεια Gain-128, και χρησιμοποιείται για την κρυπτογράφηση δεδομένων κατά τη μετάδοση τους σε κάποιο δίκτυο χρησιμοποιώντας τη μορφή κρυπτογράφησης AEAD (Authenticated Encryption and Associated Data). Η AEAD συνδυάζει τη λειτουργία κρυπτογράφησης και τον κώδικα Αυθεντικοποίησης μηνυμάτων (MAC) σε έναν αλγόριθμο καταφέρνοντας έτσι να παρέχει εμπιστευτικότητα και να διασφαλίζει την ακεραιότητα των δεδομένων που μεταδίδονται. Για την καταγραφή του κώδικα VHDL χρησιμοποιήθηκε το πρόγραμμα Notepad ++, ενώ για τη συγγραφή του κείμενου της παρούσας πτυχιακής εργασίας χρησιμοποιήθηκε το πρόγραμμα Microsoft Office 365.

Στο δεύτερο κεφάλαιο της παρούσας εργασίας, περιγράφεται η αρχιτεκτονική του αλγορίθμου Grain-128AEADv2. Αρχικά, παρουσιάζεται ο αλγόριθμος με μια σύντομη αναφορά στην έννοια IOT. Κατόπιν, ακολουθούν οι βασικές προδιαγραφές του αλγορίθμου όπου αναλύονται τα στάδια λειτουργίας του, τα βασικά blocks που τον απαρτίζουν, καθώς και ο τρόπος με τον οποίο συνδέονται τα στοιχεία του κυκλώματος μεταξύ τους. Τέλος, επισημαίνονται τα αρχεία που χρησιμοποιήθηκαν τόσο για την ανάλυση, όσο και για την προσομοίωση και περιγράφεται η λειτουργία του καθενός.

Στο τρίτο κεφάλαιο της παρούσας εργασίας περιέχονται τα αποτελέσματα της σύνθεσης και τα αποτελέσματα των λειτουργικών προσομοιώσεων από το υλικό που χρησιμοποιήθηκε. Οι προσομοιώσεις έγιναν μετά από επιτυχή σύνθεση του υλικού. Ακολουθεί σχολιασμός των αποτελεσμάτων για καλύτερη κατανόηση.

Στο τέταρτο κεφάλαιο παρατίθενται τα πλεονεκτήματα και οι περιορισμοί όσον αφορά τη χρήση του Grain-128AEADv2

Στο τέλος, υπάρχει βιβλιογραφία και ιστότοποι που χρησιμοποιήθηκαν κατά την εκπόνηση της πτυχιακής εργασίας, όσο ακόμα και τα αρχεία σε κώδικα VHDL.

Κεφάλαιο 2

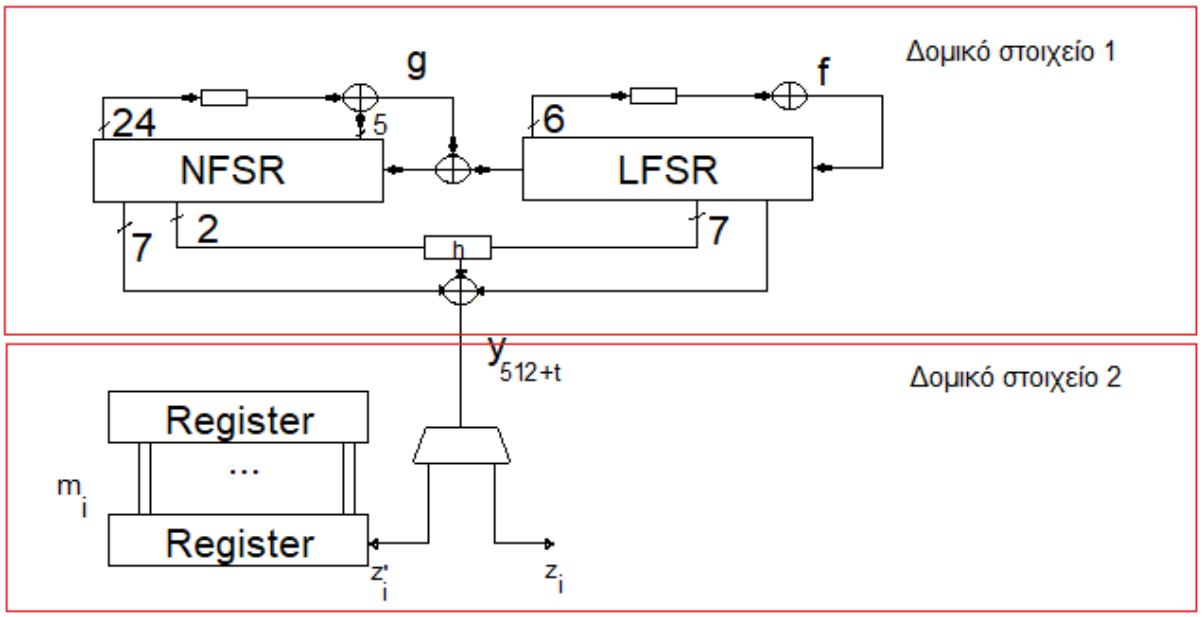
Αλγόριθμος Grain-128AEADv2

2.1. Γενική Περιγραφή

Ο Grain-128AEADv2 είναι ένας αλγόριθμος κρυπτογράφησης ροής βασισμένος στον Grain-128. Έχει σχεδιαστεί από τους Martin Hell, Thomas Johansson, Alexander Maximov, Willi Meier, Jonathan Sönnerup και Hirotaka Yoshida. Υποστηρίζει τη λειτουργεία της Αυθεντικοποίησης και επαλήθευσης δεδομένων (AEAD), επιτρέποντας έτσι στον παραλήπτη ενός μηνύματος να γνωρίζει πως τα δεδομένα που έλαβε είναι αυθεντικά και δεν έχουν τροποποιηθεί κατά τη μετάδοση τους. Ο grain-128AEADv2 χρησιμοποιείται σε εφαρμογές ασφαλείας όπως: δικτυακά συστήματα, συστήματα αποθήκευσης και μεταφοράς δεδομένων. Αποτελείται από δυο βασικά δομικά στοιχεία όπως φαίνεται και στο σχήμα 1. Το πρώτο στοιχείο είναι το «pre-output generator», το οποίο παράγει μια ροή ψευδοτυχαίων ψηφίων τα οποία χρησιμοποιούνται για την κρυπτογράφηση και τη δημιουργία ενός σήματος Αυθεντικοποίησης, το δεύτερο block επίσης παράγει ένα σήμα Αυθεντικοποίησης και στη συνέχεια αυτά τα δύο σήματα συνδυάζονται για να δημιουργήσουν το τελικό σήμα. Η σχεδιαστική του δομή μοιάζει πολύ με του Grain-128a, αλλά έχει τροποποιηθεί ώστε να υποστηρίζει τη λειτουργία AEAD, και να δέχεται μεγαλύτερο κλειδί αυθεντικοποίησης. Σε αυτό το κεφάλαιο, θα γίνει παρουσίαση του Grain-128AEADv2, των προδιαγραφών του και της οικογένειάς του, καθώς και της έννοιας IOT με σκοπό να κατανοήσουμε την ανάγκη ύπαρξης του αλγορίθμου. Στη συνέχεια, θα ακολουθήσει ο τρόπος λειτουργίας του και τέλος θα προβληθεί η λειτουργική προσομοίωση του αλγορίθμου και θα αναλυθεί η λογική στη σχεδίαση του.

2.2. Επισκόπηση Grain-128AEADv2

Ο αλγόριθμος Grain-128AEADv2 προσφέρει 128-bit ασφάλεια. Αποτελείται από δυο βασικά δομικά στοιχεία [1], το πρώτο στοιχείο είναι το «pre-output generator» το οποίο παράγει μια ροή ψευδοτυχαίων ψηφίων τα οποία χρησιμοποιούνται για την κρυπτογράφηση και τη δημιουργία ενός σήματος Αυθεντικοποίησης, το δεύτερο block επίσης παράγει ένα σήμα Αυθεντικοποίησης και στη συνέχεια αυτά τα δύο σήματα συνδυάζονται για να δημιουργήσουν το τελικό σήμα. Λειτουργεί με τον εξής τρόπο: αρχικά λαμβάνει ένα κείμενο μεταβλητού μήκους, μεταβαλλόμενου μεγέθους σχετικά δεδομένα, nonce συγκεκριμένου μεγέθους 96bits και κλειδί ορισμένου μεγέθους 128bits. Η έξοδος που παράγει είναι ένα κρυπτογραφημένο κείμενο μεταβλητού μήκους. Κατά την αποκρυπτογράφηση το κείμενο ανακτάται από ένα έγκυρα κρυπτογραφημένο μήνυμα διαφορετικά δεν επιστρέφεται τίποτα. Αλγόριθμοι όπως ο Grain-128AEADv2 προσφέρουν προστασία από επιθέσεις όπως: επίθεση ανάκτησης κλειδιού, επίθεση στο κανάλι επικοινωνίας και κάθε είδους επίθεση που σκοπό έχει την ανάκτηση του μηνύματος και την αλλοίωση αυτού.



Σχήμα 1 – Απεικόνιση δομικών στοιχείων Grain-128AEADv2

2.2.1 IoT (Internet Of Things)

Ωστόσο προτού ασχοληθούμε με το τρόπο λειτουργίας του Grain-128AEADv2 ας ανακαλύψουμε ποιο σκοπό εξυπηρετεί ο αλγόριθμος αυτός και τι ακριβώς είναι το Internet Of Things. Όταν μιλάμε για το IoT(Internet Of Things) αναφερόμαστε σε ένα δίκτυο συσκευών με φυσική υπόσταση όπως αισθητήρες, αυτοκίνητα, ή ακόμα και οικιακές συσκευές καθώς και οποιοδήποτε συσκευή, η οποία έχει ενσωματωμένο κύκλωμα και λογισμικό, όπως και την δυνατότητα σύνδεσης σε δίκτυο αποκτώντας έτσι την ικανότητα να λάβει και να στείλει πληροφορίες.

Το IoT είναι ικανό να εξελίξει πολλούς κλάδους όπως η βιομηχανία και η ιατρική λόγω της αδιάκοπης και φαινομενικά στιγμιαίας επικοινωνίας των συσκευών, βελτιώνοντας σημαντικά την απόδοση, την αυτοματοποίηση των λειτουργιών και τη παροχή στοχευμένων πληροφοριών σε πραγματικό χρόνο.

Με την αύξηση του πλήθους των συσκευών IoT, αυξάνονται σημαντικά οι δυνατότητες όπως και οι κίνδυνοι, δημιουργώντας τρωτά σημεία στην ασφάλεια του εκάστοτε δικτύου. Ως επί το πλείστον, οι συσκευές αυτές χρησιμοποιούνται σε μη ελεγχόμενα περιβάλλοντα και πολλές φορές έχουν περιορισμένη επεξεργαστική ισχύ και μνήμη. Καθίστανται, λοιπόν, ανίκανες ως προς την υλοποίηση πολύπλοκων πρωτοκόλλων ασφαλείας.

Για την αντιμετώπιση αυτών των θεμάτων ασφαλείας απαιτούνται αλγόριθμοι κρυπτογραφίας σχεδιασμένοι ειδικά για αυτές τις συσκευές. Ένας από αυτούς είναι ο Grain-128AEADv2, ο οποίος προστατεύει τα δεδομένα και εξασφαλίζει την ακεραιότητα τους κατά τη μετάδοση στο δίκτυο. Ο Grain-128AEADv2 χρησιμοποιείται τόσο για την

κρυπτογράφηση όσο και για την αποκρυπτογράφηση των δεδομένων που στέλνονται μεταξύ IoT συσκευών.

Σύμφωνα με άρθρο του FORBES τους πρώτους έξι μήνες του 2019 οι επιθέσεις αυξήθηκαν κατακόρυφα, φτάνοντας το αστρονομικό νούμερο των 2.9 δισεκατομμυρίων έναντι των 800 εκατομμυρίων για το δεύτερο μισό του 2018 [18]. Επίσης, σύμφωνα με αναφορές για τους πρώτους 2 μήνες το 2023 παρατηρήθηκε αύξηση 41% των εβδομαδιαίων επιθέσεων σε σύγκριση με το 2022 με τους περισσότερους από τους πληγέντες οργανισμούς να βρίσκονται στην Ευρώπη [19]. Η ύπαρξη, επομένως, και η χρήση των αλγορίθμων κρυπτογραφίας είναι μεγίστης σημασίας για τη προστασία των δεδομένων και για τη διατήρηση της ακεραιότητας των συστημάτων. Έχει μεγάλη αξία η χρήση ασφαλών αλγορίθμων κρυπτογραφίας κατά την ανάπτυξή και υλοποίηση IoT συστημάτων.

2.3 Προδιαγραφές Grain-128AEADv2

Σε αυτή την ενότητα θα αναλύσουμε τα δομικά στοιχεία και τις συναρτήσεις που χρησιμοποιούνται, την διαδικασία αρχικοποίησης, το τρόπο λειτουργίας και τις διαδικασίες κρυπτογράφησης και από-κρυπτογράφησης.

2.3.1 Δομικά στοιχεία και συναρτήσεις

Όπως έχουμε δει ήδη στο σχήμα 1, ο αλγόριθμος αποτελείται από δυο βασικά block: το pre-output generator και το authenticator generator. Το 1ο Μπλοκ (Pre-output generator) αποτελείται από τα εξής στοιχεία: καταχωρητής μετατόπισης γραμμικής ανάδρασης (LFSR), καταχωρητής μετατόπισης μη γραμμικής ανάδρασης (NFSR), μία συνάρτηση Pre-output. Το 2ο Μπλοκ (Authenticator generator) αποτελείται από ένα καταχωρητή (Register) και ένα συσσωρευτή (Accumulator, short term register). Στη συνέχεια, θα περιγράψω αναλυτικά το κάθε ένα από αυτά. Πρώτα, όμως, θα περιγράψω το τρόπο λειτουργίας, ο οποίος είναι ο ακόλουθος:

Αρχικά το Δομικό στοιχείο 1 παράγει μια ροή από ψευδο-τυχαία bits, τα οποία χρησιμοποιούνται για την κρυπτογράφηση και την ετικέτα ελέγχου αυθεντικότητας. Έστω ότι τα δεδομένα του 128-bit μεγέθους LFSR είναι $S_t = \left[\mathrm{S}_0^{\mathrm{t}}, \mathrm{S}_{1,\ldots}^{\mathrm{t}}, \mathrm{S}_{127}^{\mathrm{t}}\right]$ και τα δεδομένα του επίσης 128-bit μεγέθους NFSR είναι $B_t = \left[B_0^{\mathrm{t}}, B_{1,\ldots}^{\mathrm{t}}, B_{127}^{\mathrm{t}}\right]$. Αυτοί οι δύο καταχωρητές ολίσθησης αντιπροσωπεύουν 256-bit της κατάστασης του Pre-output Generator. Το πολυώνυμο ανατροφοδότησης του LFSR, που ορίζεται στο GF(2) (πεδίο δύο στοιχείων) και συμβολίζεται ως f(x), ορίζεται ως εξής:

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128}$$

Αντίστοιχα η συνάρτηση ανανέωσης του LFSR δίνεται από την εξίσωση:

$$\begin{aligned} S_{127}^{t+1} &= s_i^t + s_7^t + S_{38}^t + S_{70}^t + s_{81}^t + S_{96}^t \\ &= L(S_t) \end{aligned}$$

Το μη γραμμικό πολυώνυμο ανατροφοδότησης του NFSR, που συμβολίζεται με g(x) και ορίζεται επίσης στο GF(2), ορίζεται ως εξής:

$$g(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128} + x^{44}x^{60} + x^{61}x^{125} + x^{63}x^{67} + x^{69}x^{101} + x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117} + x^{46}x^{50}x^{58} + x^{103}x^{104}x^{106} + x^{33}x^{35}x^{36}x^{40}$$

Και η αντίστοιχη συνάρτηση ανανέωσης είναι η ακόλουθη:

$$b_{127}^{t+1} = s_0^t + b_0^t + b_{26}^t + b_{56}^t + b_{91}^t + b_{96}^t + b_3^t b_{67}^t + b_{11}^t b_{13}^t + b_{17}^t b_{18}^t + b_{27}^t b_{59}^t + b_{40}^t b_{48}^t + b_{61}^t b_{65}^t + b_{68}^t b_{84}^t + b_{22}^t b_{24}^t b_{25}^t + b_{70}^t b_{78}^t b_{82}^t + b_{88}^t b_{92}^t b_{93}^t b_{95}^t = s_0^t + F(B_t)$$

Εννέα μεταβλητές κατάστασης λαμβάνονται ως είσοδος σε μια λογική συνάρτηση h(x). Μια μεταβλητή κατάστασης ανήκει στο σύνολο των μεταβλητών που χρησιμοποιούνται για να περιγράψουν τη μαθηματική «κατάσταση» ενός δυναμικού συστήματος. Για την h(x) χρησιμοποιούνται 2-bit από την NFSR και επτά από την LFSR και ορίζεται ως εξής:

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8,$$

όπου οι μεταβλητές x_0,…..,x_8 αντιστοιχούν σε συγκεκριμένες μεταβλητές κατάστασης εντός του Grain-128AEADv2 με την εξής σειρά: $b_{12}^t, s_8^t, s_{13}^t, s_{20}^t, b_{95}^t, s_{42}^t, s_{60}^t, s_{79}^t$ και s_{94}^t. Η έξοδος του Pre-output Generator δίνεται από την συνάρτηση:

$$y_t = h(x) + s_{93}^t + \sum_{j \in A} b_j^t$$

Όπου A = {2,15,36,45,64,73,89}.

Ο Authenticator generator αποτελείται από ένα καταχωρητή ολίσθησης, ο οποίος κρατάει αποθηκευμένα τα 64 πιο πρόσφατα μονά bits της συνάρτησης Pre-output, και από έναν συσσωρευτή. Το μέγεθος και των δύο είναι 64-bits. Θέτουμε, λοιπόν, το περιεχόμενο του συσσωρευτή τη χρονική στιγμή i ως $A_i = a_0^i, a_1^i, \dots, a_{63}^i$. Παρομοίως, θέτουμε το περιεχόμενο του καταχωρητή ολίσθησης ως $R_i = r_0^i, r_1^i, \dots, r_{63}^i$.

2.3.2 Φάση αρχικοποίησης

Πρωτού η Pre-output μπορεί να χρησιμοποιηθεί ως keystream(ακολουθία bits) πρέπει να αρχικοποιηθεί η κατάσταση τόσο του Pre-output generator όσο και οι καταχωρητές του authenticator generator με ένα κλειδί και η nonce. Χαρακτηρίζω λοιπόν τα bits του κλειδιού ώς: $k_i, 0 \leq i \leq 127$ και τα bits της nonce $IV_i, 0 \leq i \leq 95$. Στη συνέχεια, η κατάσταση αρχικοποιείται ως εξής: τα 128-bits του NFSR φορτώνονται με τα bits του κλειδιού όπου $b_i^0 k_i, 0 \leq i \leq 127$ και τα πρώτα 96-bits του LFSR φορτώνονται με τα bits της nonce όπου $S_i^0 = IV_{iJ} 0 \leq i \leq 95$. Τα υπόλοιπα 32-bits του LFSR φορτώνονται με 31 άσσους και 1

μηδέν, $S_i^0 = 1, 96 \leq \mathrm{i} \leq 126,\ S_{127}^0 = 0$. Στη συνέχεια ο αλγόριθμος χρονίζεται 320 φορές, κάνοντας την πράξη XOR με τη συνάρτηση Pre-output με τη είσοδο του LFSR και του NFSR:

$$s_{127}^{t+1} = L(S_t) + y_t\,, \qquad 0 \leq t \leq 319$$

$$b_{127}^{t+1} = s_0^t + F(B_t) + S_t\,, \qquad 0 \leq t \leq 319$$

(Στο σχήμα 2 παρουσιάζεται η υλοποίηση της XOR και στο σχήμα 3 ο πίνακας αληθείας της)

Στη συνέχεια, ο αλγόριθμος χρονίζεται 64 φορές ακόμα, εισάγοντας ξανά το κλειδί, κάνοντας ξανά την πράξη XOR με το κλειδί και τις εισόδους των LFSR & NFSR αντίστοιχα:

$$s_{127}^{t+1} = L(S_t) + K_{t-256}\,, \qquad 320 \leq t \leq 383$$

$$b_{127}^{t+1} = s_0^t + F(B_t) + K_{t-320}\,, \qquad 320 \leq t \leq 383$$

Εφόσον ολοκληρωθεί η διαδικασία αρχικοποίησης του Pre-output Generator, ξεκινά η αρχικοποίηση του Authenticator generator φορτώνοντας τον καταχωρητή και τον συσσωρευτή με τον εξής τρόπο:

$$\alpha_j^0 = y_{384+j}, \qquad O \leq j \leq 63$$

$$r_j^0 = y_{448+j}, \qquad O \leq j \leq 63$$

Αφού αρχικοποιηθούν ο καταχωρητής και ο συσσωρευτής ο LFSR ανανεώνεται στιγμιαία:

$$s_{127}^{t+1} = L(S_t)\,, \qquad 384 \leq t \leq 511$$

Και ο NFSR ανανεώνεται επίσης:

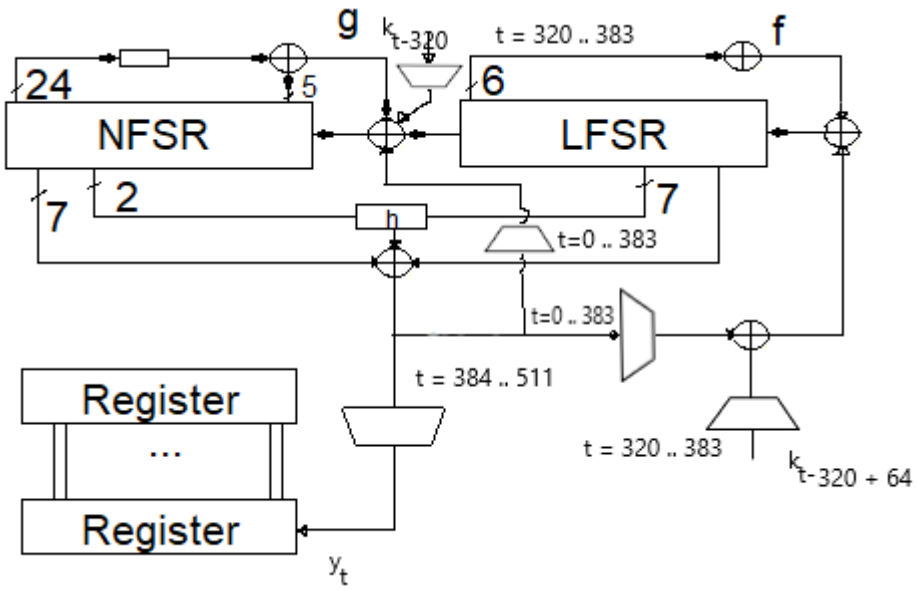
$$b_{127}^{t+1} = s_0^t + F(B_t) +\,, \qquad 384 \leq t \leq 511$$

Τέλος όταν ο αλγόριθμος έχει αρχικοποιηθεί πλήρως η κατάσταση των LFSR & NFSR δίνονται από τα S_{512} και B_{512} αντίστοιχα η κατάσταση του καταχωρητή και του συσσωρευτή δίνεται από τα R_0 και A_0. Η διαδικασία της αρχικοποιήσης παρουσιάζεται στο σχήμα 4.

Σχήμα 2 – Υλοποίηση της XOR με λογικές πύλες

Input		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Σχήμα 3 – Πίνακας αληθείας της XOR



Σχήμα 4 – Παρουσίαση της αρχικοποίησης του Grain-128AEADv2

2.3.3 Παρουσίαση λειτουργίας

Έστω πως έχουμε μήνυμα m με μήκος L, άρα $m_0, m_{2,\dots}, m_{L-1}$, θέτουμε $m_L = 1$ ώστε να διασφαλίσουμε πως m και m||0 έχουν διαφορετικές τιμές. Εφόσον, αρχικοποιηθεί το Pre-output generator, χρησιμοποιείται για να δημιουργήσει το keystream με Z_i bits για κρυπτογράφηση και Z_i' για την αυθεντικοποίηση για την ενημέρωση του καταχωρητή του authenticator generator.

$$Z_i = y_{512+2i},$$

Για παράδειγμα, κάθε ζυγό bit (μετρώντας από το μηδέν) από το Pre-output generator χρησιμοποιείται ως keystream bit. Τα bits αυθεντικοποίησης δημιουργούνται ως εξής:

$$Z_i' = y_{512+2i+1}$$

Άρα κάθε μονό bit από το pre-output generator λογίζεται ως bit αυθεντικοποίησης. Συνεχίζοντας, το μήνυμα κρυπτογραφείται ως εξής:

$$c_i = m_i \oplus z_i\,, \qquad 0 \le \mathrm{i} < L$$

Ο συσσωρευτής ανανεώνεται ως εξής:

$$a_j^{i+1} = a_j^i + m_i\, r_j^i, \qquad 0 \le j \le 63, \qquad 0 \le \mathrm{i} < L$$

Και τέλος ο καταχωρητής ολίσθησης ανανεώνεται όπως φαίνεται παρακάτω:

$$r_{63}^{i+1} = z_i',$$

$$r_j^{i+1} = a_{j+1}^i, \qquad 0 \le j \le 62$$

2.3.4 Περιορισμοί ροής κλειδιού

Οι αλγόριθμοι της οικογένειας Grain έχουν σχεδιαστεί να υποστηρίζουν την κρυπτογράφηση μεγάλων κομματιών δεδομένων χρησιμοποιώντας το ίδιο ζευγάρι κλειδιού και nonce. Ο Grain-128AEADv2, ωστόσο, περιορίζει τον αριθμό των keystream bits για κάθε ζεύγος key/nonce σε 280, επομένως ο αριθμός των pre-output bits που μπορούν να παραχθούν για ένα ζεύγος είναι 281.

2.3.5 Authenticated Encryption with Associated Data (AEAD)

Το μοντέλο κρυπτογράφησης AEAD παρέχει τόσο εμπιστευτικότητα όσο και ακεραιότητα για το κρυπτογραφημένο κείμενο, μπορεί εν τέλει να κρυπτογραφήσει το κείμενο και να εγγυηθεί για την ακεραιότητα του «αντικειμένου» που κρυπτογραφεί. Παρ' όλ' αυτά υποστηρίζει την πιστοποίηση της ακεραιότητας των δεδομένων, τα οποία δεν είναι κρυπτογραφημένα. Ο Grain-128AEADv2 το επιτυγχάνει αυτό δίνοντας τη τιμή μηδέν σε ορισμένα bits ($y_{512} + 2i$) τα οποία δε θα κρυπτογραφηθούν αλλά θα πρέπει να επαληθευτούν. Στην ουσία μπορεί να ελέγχει σε επίπεδο bit τα σχετικά δεδομένα και μπορούν να εμφανιστούν οπουδήποτε μέσα στο μήνυμα.

Ας δούμε λοιπόν τι ακριβώς σημαίνει αυτό. Θα φτιάξουμε μια AEAD mask με την οποία μπορεί κάποιος να ελέγξει σε ένα μήνυμα ποιο κομμάτι του θα κρυπτογραφηθεί και πιστοποιηθεί. Η μάσκα αυτή μπορεί να ελέγξει σε κείμενο ποια bit θα κρυπτογραφηθούν και ποια μόνο θα αυθεντικοποιηθούν. Τα bits τα οποία μόνο αυθεντικοποιούνται ονομάζονται Associated Data (σχετικά δεδομένα). Η μάσκα AEAD παράγεται συνήθως από την nonce και ο σκοπός της είναι να εξασφαλίσει πως οι λειτουργίες κρυπτογράφησης και αυθεντικοποίησης εκτελούνται μόνο στα κατάλληλα bits του μηνύματος. Θα θέσουμε:

$$d = d_0, d_1, \dots, d_{L-1}$$

Με το τρόπο αυτό δηλώνουμε τα bit που θα κρυπτογραφηθούν. Αν η μάσκα περιέχει μόνο άσσους (δηλαδή όχι σχετικά δεδομένα), η κρυπτογράφηση γίνεται όπως φαίνεται στο σχήμα 5, ενώ αν η μάσκα περιέχει μηδενικά γίνεται με τον ακόλουθο τρόπο:

$$c_i = m_i \oplus z_i * d_i, \qquad 0 \le i < L$$

και ανάλογα γίνεται και η αποκρυπτογράφηση. Η μάσκα AEAD πρέπει να είναι προκαθορισμένη για το πρωτόκολλο που χρησιμοποιεί ο Grain-128AEADv2. Η χρήση μιας τέτοιας παρέχει μεγάλη ευελιξία, καθώς επιτρέπει την ύπαρξη μη κρυπτογραφημένων δεδομένων, όχι μόνο στο αρχικό μέρος ενός πακέτου δυαδικών ψηφίων, αλλά μπορούν να υπάρχουν μη κρυπτογραφημένα δεδομένα σε οποιαδήποτε θέση μέσα στο πακέτο. Με άλλα λόγια, προσφέρει μεγάλη ευελιξία στη σχεδίαση του πρωτοκόλλου. Γεννιέται έτσι το μειονέκτημα πως ο αλγόριθμος παράγει κάποια bits τα οποία δεν μπορούν να

χρησιμοποιηθούν ούτε στην κρυπτογράφηση αλλά ούτε στην αυθεντικοποίηση. Ωστόσο, αφού δεν υπάρχει οποιοδήποτε επιπλέον κόστος στο να συμπεριληφθούν μη-κρυπτογραφημένα δεδομένα στην κρυπτογράφηση θεωρείται αποτελεσματική λύση.

Όταν χρησιμοποιούμε μεταβλητού μήκους σχετικά δεδομένα πρέπει να δώσουμε ιδιαίτερη σημασία. Αν θέλουμε να αφήσουμε τα πρώτα X bits του κειμένου χωρίς κρυπτογράφηση, και να κρυπτογραφήσουμε τα υπόλοιπα (x-L bits), μπορούμε να πούμε ότι $d_i = \boldsymbol{O}$, αν $i < \boldsymbol{x}$ το bit δεν είναι κρυπτογραφημένο ενώ αν $\boldsymbol{d_i = 1}$ είναι κρυπτογραφημένο. Η παραπάνω συνθήκη μας βοηθάει να κρυπτογραφήσουμε το κείμενο. Εάν έχουμε κρυπτοκείμενο (ad, μήκος ad X, μήνυμα, MAC) με μορφή:

$$\left((m_0, m_1, \dots, m_{x-1}), x, (m_x, m_{x+1}, \dots), t\right)$$

Και αντικατασταθεί με κρυπτοκείμενο της μορφής:

$$\left((m_0, m_1, \dots, m_{x-1}, m_\chi), x+1, (m_{x+1}, m_{x+2}, \dots), t\right)$$

Καταλήγουμε στο συμπέρασμα πως όταν έχουμε μεταβλητού μήκους σχετικά δεδομένα, είναι σημαντικό να διασφαλιστεί ότι οποιεσδήποτε δυνητικές επιθέσεις σε επίπεδο πρωτοκόλλου αντιμετωπίζονται κατάλληλα, κάτι που ισχύει καθώς από το παράδειγμα διαφαίνεται πως δεν αναιρείται η εγκυρότητα του MAC καθώς παράγεται ανεξάρτητα από αυτές τις μεταβολές.

2.3.6 Χρήση Grain-128AEADv2 με το API του NIST

Σε αυτό το κεφάλαιο θα περιγράφει η λειτουργία του Grain-128AEADv2 βάσει του API (Application Programming Interface) του NIST, η οποία είναι μια διεπαφή από το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας για την υλοποίηση κρυπτογραφικών πρωτοκόλλων. Αρχικά θα καθοριστεί το τι συμβολίζει το γράμμα, m είναι το μήνυμα που θα σταλεί στο API και θα κρυπτογραφηθεί, m’ υποδηλώνει τη πλήρη συμβολοσειρά που χρησιμοποιείται από το πυρήνα του αλγορίθμου. Αντίστοιχα, το c είναι το κρυπτοκείμενο, περιλαμβάνει τη κρυπτογραφημένη συμβολοσειρά και τις πληροφορίες επαλήθευσης ταυτότητας, c’ είναι η συμβολοσειρά που αποκρυπτογραφείται και περιλαμβάνει και τα σχετικά δεδομένα. Αξίζει να σημειωθεί ότι το API του NIST “byte oriented”, που σημαίνει ότι για να πραγματοποιηθούν οι λειτουργίες τα δεδομένα χωρίζονται τα οποία είναι χωρισμένα σε bytes. Αυτή η ιδιαιτερότητα του συγκεκριμένου API δεν θέτει σημαντικούς περιορισμούς στον αλγόριθμο αλλά το bit που ορίζει το αρχικό κείμενο αντί για ‘1’ θα είναι 0x80, όπου λόγω του σχεδιασμού του αυθεντικοποιητή είναι ισοδύναμες.

2.3.6.1 Κρυπτογράφηση

Για τη συγκεκριμένη περίπτωση του NIST λογισμικού API, προτείνουμε να χαρτογραφήσουμε την είσοδο σε bytes (ad, ad length, message, message length) σε μια συμβολοσειρά m' με τον τρόπο που ακολουθεί.

$$m' = Encode(ad\ length)||ad||m||0x80$$

Όπου Encode() = y είναι η συνάρτηση με την οποία ένας ακέραιος αριθμός μετατρέπεται στην αντίστοιχα δυαδικά ακολουθία με συγκεκριμένο μήκος. Αν το πρώτο bit του y είναι 0, τότε τα υπόλοιπα 7 bytes αντιπροσωπεύουν έναν κωδικό για τον αριθμό των bytes στα σχετιζόμενα δεδομένα (έως και 127 bytes). Αντίθετα, εάν το πρώτο byte στον αριθμό y ξεκινά με 1, τα υπόλοιπα 7 bits αντιπροσωπεύουν έναν κωδικό για τον αριθμό των bytes που ακολουθούν για να περιγράψουν το μήκος (σε bytes) των σχετικών δεδομένων. Στη προκειμένη περίπτωση το πρώτο byte του y, ακολουθείται από τα bytes που περιγράφουν το μήκος. Στην συνέχεια, κρυπτογραφείται και αυθεντικοποιείται το m' με το μοντέλο AEAD με χρήση της μάσκας $d_i = 0$ για όλα τα bits που ισχύει $i < M$, όπου το M υποδηλώνει το μήκος της συμβολοσειράς $(ad\ length)||ad$. Αντίστοιχα για $i \geq M$, $d_i = 1$. Η περίληψη της AEAD κρυπτογράφησης του αλγορίθμου με χρήση του API του NIST δίνεται στο σχήμα 6

Input: ad, adlen, m, mlen, k, nonce
Output: c

1. Initialize generator with k and nonce
2. Construct m' = (Encode(adlen)||ad||m||0x80)
3. Let
4. M = bit length of Encode(adlen)||ad
5. $d_i = 0$, $(0 \leq i \leq M - 1)$
6. $d_i = 1$, $(M \leq i \leq M + mlen - 1)$
7. **Encrypt** using $c'_i = m'_i \oplus z_i d_i$, $(0 \leq i \leq M + mlen - 1)$
8. **Authenticate** using z'_I and generate $A_{M+mlen+1}$
9. $c = (c'_M, c'_{M+1}, \ldots, c'_{M+mlen-1})||A_{M+mlen+1}$
10. return 0

Σχήμα 5 – AEAD Encrypt with NIST API

2.3.6.2 Αποκρυπτογράφηση

Στην πλευρά του παραλήπτη αρχικά πρέπει να επαληθευτή ο MAC (Message Authentication Code) και να εκτελεστεί η αποκρυπτογράφηση. Το API παίρνει ως είσοδο τα σχετικά δεδομένα και το κρυπτοκείμενο c το οποίο αποτελείται από το κρυπτογραφημένο μήνυμα και το MAC. Το μήκος των σχετικών δεδομένων είναι κωδικοποιημένο, και προστίθεται το "padding" για να διαμορφωθεί το c'. Το m' υπολογίζται ως εξής:

$$m'_i = c'_i \oplus z_i \cdot d_i, \qquad 0 \leq i < L$$

Σημαντικό βήμα της αποκρυπτογράφησης είναι η σύγκριση του MAC που συνόδευσε το μήνυμα με το MAC που υπολογίστηκε από το m'. Αφού συγκριθούν τα 64-bits MAC δημιουργείται ένας δείκτης γνωστός ως "flag" όπου αν πάρει τη τιμή 0 τα MAC είναι ίδια, διαφορετικά παίρνει τη τιμή -1. Η σύγκριση αυτή αποτελεί βασικό στοιχείο της αποκρυπτογράφησης. Η περίληψη της AEAD αποκρυπτογράφησης του αλγορίθμου με χρήση του API του NIST δίνεται στο σχήμα 7. Επίσης να σημειωθεί πως το "padding" είναι 0x80, το οποίο οφείλεται στη "byte oriented" φύση του API.

Input: ad, adlen, m, mlen, k, nonce
Output: m

1. Initialize generator with k and nonce
2. Construct m' = (Encode(adlen)||ad||c_0,…,$c_{clen-65}$ ||0x80)
3. Let
4. M = bit length of Encode(adlen)||ad
5. $d_i = 0$, $(0 \leq i \leq M - 1)$
6. $d_i = 1$, $(M \leq i \leq M + mlen - 1)$
7. **Dencrypt** using $m'_i = c'_i \oplus z_i d_i$, $(0 \leq i \leq M + mlen - 1)$
8. **Authenticate** using z'_i and generate $A_{M+mlen+1}$
9. Set m = m'_M,…, $m'_{M+mlen-1}$
10. **If (**$c_{clen-64}$,…, c'_{clen-1}) == $A_{M+mlen+1}$
11. return 0
12. **else**
13. return -1

Σχήμα 6 – AEAD Decrypt with NIST API

2.4 Σχεδιαστική λογική Grain-128AEADv2

Αυτή η ενότητα παρουσιάζει σύντομα την οικογένεια των αλγορίθμων Grain και πως ο σχεδιασμός τους έχει εξελιχθεί μέχρι σήμερα. Επιπλέον, θα γίνει αναφορά ανάμεσα στις διαφορές του Grain-128a και του υπό εξέταση για αυτήν πτυχιακή Grain-128AEADv2.

2.4.1 Ιστορική αναδρομή της οικογένειας Grain

Η οικογένεια των αλγορίθμων κρυπτογράφησης ροής Grain βασίζεται στην ιδέα πίσω από το Nonlinear Filter generator (NLF) . Σε ένα μη γραμμικό φίλτρο, ο LFSR χρησιμοποιείται για να παράσχει μια ακολουθία με μεγάλη περίοδο, και μια μη γραμμική συνάρτηση, που

παίρνει μέρη της ακολουθίας του LFSR ως είσοδο, χρησιμοποιείται για να προσθέσει μη γραμμικότητα στην ακολουθία των κλειδιών. Η ασφάλεια ενός μη NLF εξαρτάται από τις ιδιότητες της μη γραμμικής συνάρτησης που χρησιμοποιείται. Ωστόσο, έχει αποδειχθεί ότι η σχεδίαση ενός ασφαλούς NLF με λογικό μέγεθος υλικού είναι πολύ δύσκολη διαδικασία, καθώς υπάρχουν πολλά εμπόδια που μπορούν να οδηγήσουν σε κενά ασφαλείας. Για το λόγο αυτό, έχει γίνει πολύς κόπος στην ανάλυση και βελτίωση του σχεδιασμού NLF. Συγκεκριμένα, έχει αποδειχθεί ότι οι «αλγεβρικές επιθέσεις» είναι πολύ αποτελεσματικές κατά του σχεδιασμού με χρήση NLF. Οι αλγεβρικές επιθέσεις βασίζονται στη χρήση αλγεβρικών εξισώσεων για να εξάγουν πληροφορίες για το κρυπτοσύστημα. Αυτές οι επιθέσεις έχουν επιτυχία κατά των μη γραμμικών γεννητριών φίλτρου, όπως οι συναρτήσεις στην οικογένεια Grain, διότι μπορούν να αποκαλύψουν τις αλγεβρικές σχέσεις μεταξύ των μεταβλητών που χρησιμοποιούνται. Αυτό μπορεί να αποκαλύψει κλειδιά και να επιτρέψει στον επιτιθέμενο να αποκρυπτογραφήσει τα δεδομένα [6] [7].

Ο grain είναι ένας αλγόριθμος κρυπτογραφίας ροής που χρησιμοποιεί ένα συνδυασμό LFSR και NFSR για τη δημιουργία μιας ροής κλειδιών. Για να ενισχυθεί η ασφάλεια του αλγορίθμου έναντι αλγεβρικών επιθέσεων προστέθηκε ένα επιπλέον NFSR στο μη γραμμικό συνδυαστή. Η αρχική έκδοση του αλγορίθμου υποβλήθηκε σε κρυπτανάλυση και έτσι αναδείχθηκε η ανάγκη για μεγαλύτερη αντοχή και μη γραμμικότητα στις μη γραμμικές συναρτήσεις, ώστε να αυξηθεί η άμυνά του σε αλγεβρικές επιθέσεις. Αυτό είχε ως αποτέλεσμα την δημιουργία μιας τροποποιημένης έκδοσης γνωστή ως Grain v1 [8], η οποία επιλέχθηκε για το τελικό κατάλογο του eSTREAM. Η Grain v1 χρησιμοποιεί ένα κλειδί 80 bit, ενώ προτάθηκε μια έκδοση με 128-bit κλειδί [9]. Παρότι οι αρχικές εκδόσεις του Grain υπέστησαν ανάλυση ασφαλείας και κρυπτανάλυση από μερικούς ερευνητές, το σχέδιο βελτιώθηκε και εξελίχθηκε σε διαφορετικές εκδόσεις με την πάροδο του χρόνου. Η Grain-128 σχεδιάστηκε πιο αποτελεσματικά από την αρχική έκδοση του Grain, αλλά η επίθεση κρυπτανάλυσης αποκάλυψε κάποιες αδυναμίες στο σχεδιασμό του [11]. Η Grain-128a πρότεινε τη χρήση ενός συναρτησιακά πλούσιου σχήματος παραγωγής και πρόσθεσε μια προαιρετική λειτουργία αυθεντικοποίησης. Παρά τις βελτιώσεις στον σχεδιασμό του Grain-128, η έκδοση αυτή δεν θεωρείται σήμερα ασφαλής και δεν πρέπει να χρησιμοποιείται. [10] [12].

Το σχέδιο που προτάθηκε για τους Γύρους 1 και 2 της διαδικασίας lightweight cryptography του NIST, ο Grain-128AEAD, βασίζεται στενά στον Grain-128a, χρησιμοποιώντας τις ίδιες συναρτήσεις ανατροφοδότησης και εξόδου. Ωστόσο, έχουν γίνει μικρές τροποποιήσεις για να προστεθεί ασφάλεια και να μεγιστοποιήσει την ανθεκτικότητα του σε επίθεσεις γρήγορης συσχέτισης (Fast correlation attack) [14]. Το σχέδιο που προτάθηκε για τον 3ο γύρο της διαδικασίας, ο Grain-128AEADv2, βασίζεται στο Grain-128AEAD. Προστίθεται μια αλλαγή στο τμήμα αρχικοποίησης για να προστατευτεί από επίθεση ανακατασκευής του κλειδιού από μια γνωστή κατάσταση όπως προτείνεται στο [15].

2.4.2 Grain-128AEADv2: Η αναβάθμιση

Όπως στους περισσότερους αλγορίθμους ροής, στο σχεδιασμό του Grain παρατηρείται μεγάλη έμφαση στις επιθέσεις γνωστές ως «State recovery attacks after initialization». Στις επιθέσεις αυτές ο επιτιθέμενος προσπαθεί να ανακτήσει τη κατάσταση του συστήματος αφού αυτό έχει αρχικοποιηθεί χρησιμοποιώντας μια γνωστή κατάσταση. Το νέο μοντέλο του Grain, υιοθετεί μια στρατηγική άμυνας που προσφέρει μέτρα προστασίας ενάντια σε τέτοιου είδους επιθέσεις αλλά και για πιο ισχυρές επιθέσεις. Με άλλα λόγια ο σκοπός είναι να παρέχεται ολοκληρωμένη και αποτελεσματική προστασία ενάντια σε κάθε απειλή.

Ο νέος στόχος ασφαλείας που τίθεται για τον Grain-128AEADv2 είναι η μείωση της δυνατότητας του επιτιθέμενου να ανακατασκευάσει το κλειδί βασιζόμενος στην κατάσταση που γνωρίζει. Αυτό επιτυγχάνεται με προσθήκες στην φάση της αρχικοποίησης.

Οι ιδιότητες του Grain-128AEADv2 είναι:

- Ο αλγόριθμος κατά την αρχικοποίηση του χρονίζεται 320 φορές, πολύ περισσότερες από τις 64 της προηγούμενης έκδοσης του. Έτσι, όλες οι αναλύσεις που έχουν γίνει σε αυτό το κομμάτι συνεχίζουν να ισχύουν.
- Χρονίζεται επιπλέον 64 φορές για την επαναφορά του κλειδιού, πρωτού οι καταχωρητές A και R αρχικοποιηθούν σε 128 κύκλους ρολογιού.
- Ο καταχωρητής και ο συσσωρευτής αρχικοποιούνται μετά την επαναφορά του κλειδιού, άρα όποια κρυπτανάλυση είχε πραγματοποιηθεί σε αυτούς δεν ισχύει πλέον.
- Τέλος, με την αύξηση του χρόνου αρχικοποίησης κατά 33%($\approx 100 * (512/384 - 1)$), αναμένεται πως ο Grain-128AEADv2 θα προσφέρει ένα καλό συμβιβασμό ανάμεσα στην ασφάλεια και την απόδοση υλικού.

2.4.3 Σύγκριση Grain-128a & Grain-128AEADv2

Ο Grain-128AEADv2 βασίζεται στον Grain-128a. Ωστόσο, φέρει αρκετές τροποποιήσεις οι περισσότερες εκ των οποίων προστέθηκαν κατά κύριο λόγο για τον διαγωνισμό του NIST, αλλά και από τις πρόσφατες ανακαλύψεις στον τομέα της κρυπτογραφίας.

Ο συσσωρευτής και ο καταχωρητής έχουν αυξηθεί στα 64-bit από τα 32-bit για μπορούν δημιουργήσουν ετικέτες αυθεντικοποίησης (MAC) μεγαλύτερου μεγέθους (64-bit).

Ο Grain-128a υποστήριζε λειτουργία κρυπτογράφησης δίχως αυθεντικοποίηση. Αυτό του έδινε τη δυνατότητα να λειτουργήσει με «λιγότερο» υλικό καθώς εξαλειφόταν η ανάγκη δύο επιπλέον καταχωρητών και η σχετική λογική τους θα μπορούσε να παραλειφθεί από την υλοποίηση του συστήματος. Επιπλέον, η λειτουργία αυτή ήταν πιο αποδοτική καθώς δεν περιλάμβανε την αρχικοποίηση του καταχωρητή και του συσσωρευτή και κάθε pre-output bit χρησιμοποιούταν ως ροή κλειδιού. Ο προτεινόμενος Grain-128AEADv2 είναι ξεκάθαρα ένας αλγόριθμος αυθεντικοποίησης και κρυπτογράφησης και η αυθεντικοποίηση των δεδομένων υποστηρίζεται πάντα. Έτσι, υπάρχει μόνο ένας τρόπος λειτουργίας.

Ο Grain-128AEADv2 επανεισάγει το κλειδί κατά την αρχικοποίηση σε κάθε κύκλο ρολογιού. Συγκεκριμένα, τα πρώτα 64-bit του κλειδιού μετακινούνται σειριακά στον NFSR ενώ τα υπόλοιπα 64 μετακινούνται σειριακά στον LFSR. Αυτό αποτελεί ένα συμβιβασμό μεταξύ παράλληλης ταχύτητας και εξάρτησης υλικού καθώς είναι μια πιο χρονοβόρα διαδικασία αλλά απαιτεί λιγότερο hardware για την υλοποίηση.

Τόσο ο Grain-128AEAD όσο και ο Grain-128a, χρειάζονται 256 κύκλους ρολογιού, πριν αρχικοποιήσουν τους καταχωρητές A και R και την ταυτόχρονη επανεισαγάγει του κλειδιού για τον Grain-128AEAD. Ο Grain-128AEADv2 έχει 320 κύκλους. Έπειτα, ο αλγόριθμος επανεισαγάγει το κλειδί, ακολουθούμενο από την αρχικοποίηση του συσσωρευτή και του καταχωρητή. Το Grain-182AEADv2 ακολουθεί μια διαδικασία αρχικοποίησης πριν ξεκινήσει η παραγωγή του keystream, η οποία απαιτεί 512 κύκλους ρολογιού. Αυτό επιτυγχάνεται μέσω της επαναφοράς του κλειδιού στο εσωτερικό κατά τη διάρκεια των κύκλων αρχικοποίησης. Αυτή η διαδικασία αρχικοποίησης εξασφαλίζει ότι ο Grain-182AEADv2 προσφέρει το ίδιο επίπεδο ασφάλειας με τις προηγούμενες εκδόσεις του, αλλά απαιτεί περισσότερους κύκλους ρολογιού για την ολοκλήρωση της αρχικοποίησης. Παρόλο που η ανάκτηση της κατάστασης θα θεωρούνταν παραβίαση του συστήματος, το αντίκτυπο στις συσκευές που έχουν εφαρμόζουν τον Grain-182AEADv2 θα είναι περιορισμένος, καθώς η ανάκτηση της κατάστασης θα επηρέαζε μόνο το μήνυμα που είναι σε εξέλιξη και όχι όλα τα μηνύματα που χρησιμοποιούν το ίδιο κλειδί.

Οι κρυπταλγόριθμοι ροής Grain έχουν σχεδιαστεί για να επιτρέπουν την κρυπτογράφηση μεγάλου όγκου δεδομένων χρησιμοποιώντας το ίδιο ζεύγος κλειδιού / nonce. Προηγουμένως, οι αλγόριθμοι της οικογένειας Grain δεν είχαν κανένα περιορισμό για το μήκος του keystream. Ωστόσο, ο σχεδιασμός του Grain-128AEADv2 έχει περιορίσει τον αριθμό των bit της ροής κλειδιού για κάθε ζεύγος κλειδιού / nonce σε 2^{80}. Εικάζεται ότι αυτό το μέγεθος είναι πολύ περισσότερο από αυτό που θα χρειαστεί στο μέλλον. Η περιοριστική συνθήκη στο μήκος της ροής κλειδιού έχει ως σκοπό να καταστήσει πιο δύσκολες τις επιθέσεις που χρησιμοποιούν γραμμικές προσεγγίσεις.

2.4.4 Επιλογές σχεδίασης για κάθε Block

Ο Grain-128AEADv2 αποτελείται από κάποια δομικά στοιχεία (Blocks). Αυτά τα Blocks συνεργάζονται μεταξύ τους για να παρέχουν στο εκάστοτε σύστημα την απαραίτητη ασφάλεια. Σε αυτή την ενότητα θα εξεταστούν αυτά τα blocks, όσον αφορά το τρόπο λειτουργίας τους, τις ιδιαιτερότητες που μπορεί να έχουν καθώς και η ανθεκτικότητα τους σε επιθέσεις. Μέσα από αυτή την ανάλυση θα καταστεί κατανοητό πως ο Grain-128AEADv2 έχει σχεδιαστεί με βασικό γνώμονα την ασφάλεια και πως κάθε ένα από τα Block που τον απαρτίζουν έχουν επιλεχτεί πολύ προσεκτικά για αυτό το σκοπό.

2.4.4.1 LFSR & NFSR

Ο αλγόριθμος Grain-128a χρησιμοποιεί κλειδί μεγέθους 128-bit. Λόγω επιθέσεων time/memory/data tradeoff, η συνήθης στρατηγική είναι το μέγεθος εσωτερικής κατάστασης του αλγορίθμου να είναι τουλάχιστον 2 φορές μεγαλύτερο από το κλειδί. Επιβάλλοντας περιορισμούς στη ροή κλειδιού θα ήταν εφικτό να περιοριστεί αυτή η απαίτηση. Ωστόσο, για το αν θα έπρεπε να αλλάξει η σχεδίαση των καταχωρητών που χρησιμοποιούνται, θα κρινόταν απαραίτητο να δημιουργηθούν και νέες συναρτήσεις. Μια τόσο σημαντική αλλαγή στο τρόπο λειτουργίας θα σήμαινε πως τα αποτελέσματα όσων ερευνών έχουν πραγματοποιηθεί στους αλγορίθμους Grain είναι ανούσια και άρα δεν μπορούν ληφθούν αποφάσεις βασισμένες σε αυτές.

2.4.4.2 Αύξηση της ταχύτητας.

Η έννοια “throughput” αναφέρεται στην ταχύτητα με την οποία ο αλγόριθμος κρυπτογραφεί και αποκρυπτογραφεί δεδομένα. Η ταχύτητα μπορεί να αυξηθεί αν προστεθούν περισσότερα αντίγραφα των boolean συναρτήσεων f,g και h έως και 32 φορές. Για να απλοποιηθεί αυτή η υλοποίηση οι συναρτήσεις αυτές δεν χρησιμοποιούν τα 31 λιγότερα σημαντικά bits των LFSR και NFSR, έτσι επιτυγχάνεται αύξηση της ταχύτητας χωρίς την ανάγκη να μεγαλώσουν σε μέγεθος τα LFSR και NFSR. Το επιπλέον υλικό δεν οδηγεί απαραίτητα σε γραμμική αύξηση της ταχύτητας όταν το σύστημα λειτουργεί στη μέγιστη συχνότητα, που σημαίνει πως η αύξηση της ταχύτητας γίνεται πιο αισθητή στις χαμηλότερες συχνότητες λειτουργίας.

2.4.4.3 Συναρτήσεις f, g, Pre-output

Η συνάρτηση f είναι μια μη γραμμική συνάρτηση, η οποία δέχεται 64-bit είσοδο και δίνει 64-bit αποτέλεσμα. Η συνάρτηση του LFSR πρέπει να πληροί 2 βασικές προϋποθέσεις: 1ον πρέπει να είναι πρωτογενής και 2ον πρέπει να έχει αρκετά “Taps” θέσεις στις οποίες το bit εξόδου συνδυάζεται με προηγούμενα bits με τη πράξη XOR για να δημιουργήσουν το επόμενο bit. Έχοντας αρκετές τέτοιες θέσεις μπορεί να αντισταθεί σε επιθέσεις συσχέτισης.

Η συνάρτηση του NFSR είναι η g, και συσχετίζει μη-γραμμικά τα bit της κατάστασης του αλγορίθμου κάνοντας έτσι την ανάλυση του αλγορίθμου από κάποιον “attacker” πολύ πιο δύσκολη. Πρέπει να είναι μεγάλου βαθμού συνάρτηση ώστε να παρουσιάζει αντίσταση σε “Cube Attacks”, επιθέσεις οι οποίες ανήκουν στην κατηγορία των αλγεβρικών επιθέσεων και εκμεταλλευόμενες κάποιες ιδιότητες της εξίσωσης μπορούν να σπάσουν τον αλγόριθμο. Πρέπει επίσης η g να παρουσιάζει υψηλά επίπεδα μη γραμμικότητας και ανθεκτικότητας σε επιθέσεις συσχέτισης καταστάσεων. Όπως αποδείχτηκε στο επιστημονικό άρθρο των Itai Dinur, Tim G¨uneysu, Christof Paar, Adi Shamir και Ralf Zimmermann [16] δεν αρκεί μια συνάρτηση 2ου βαθμού όπως στον Grain-128, επομένως στον Grain-128AEADv2 η συνάρτηση είναι 4ου βαθμού όπως και στον Grain-128a. Η συνάρτηση b απεικονίζεται στο σχήμα 8, και έχει μη γραμμικότητα 8356352 και παρόλο που θεωρείται υψηλή τιμή μη

γραμμικότητας δεν είναι ο μόνος παράγοντας που έχει σημασία όσον αφορά την ανθεκτικότητα μιας συνάρτησης.

$$b(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_8x_9 + x_{10}x_{11} + x_{12}x_{13} + x_{14}x_{15}x_{16} + x_{17}x_{18}x_{19} + x_{20}x_{21}x_{22}x_{23},$$

Σχήμα 7 - Συνάρτηση b(x) του Grain-128

Η ανθεκτικότητα της g ενισχύεται με τη προσθήκη 5 γραμμικών όρων επιτυγχάνοντας έτσι μεγαλύτερη πολυπλοκότητα με τη μη γραμμικότητά της να διαμορφώνεται ως εξής:

$2^5 * 8356352 = 267403264$ και Resiliency (Ανθεκτικότητα) 4. Υπάρχουν 2^{14} γραμμικές προσεγγίσεις της g(x) με παράγοντα απόκλισης 2^{-15} και μέγιστη απόκλιση 2^{-9}. Αυτές οι γραμμικές προσεγγίσεις χρησιμοποιούνται για να αναλυθεί η συμπεριφορά και οι ιδιότητες της συνάρτησης, ειδικά όσον αφορά τη γραμμικότητά της και την ικανότητά της να αντιστέκεται σε συγκεκριμένους τύπους επιθέσεων.

Η συνάρτηση Pre-output δέχεται είσοδο από το LFSR και NFSR, αντίστοιχα με τη g η Pre-output αποτελείται από τη μη γραμμική h και ένα γραμμικό κομμάτι. Η μη-γραμμική h έχει μη γραμμικότητα 240 και προσθέτει 8 γραμμικούς όρους με τη μη γραμμικότητα της να διαμορφώνεται ως εξής: $2^8 * 240 = 61440$. Υπάρχουν συνολικά 2^8 γραμμικές προσεγγίσεις με μέγιστο παράγοντα απόκλισης 2^{-5}.

2.4.4.4 Αυθεντικοποίηση Μηνυμάτων

Ο μηχανισμός αυθεντικοποίησης μηνυμάτων του Grain-128AEADv2 βασίζεται σε καθολικές συναρτήσεις κατακερματισμού. Ο αποστολέας και ο παραλήπτης συμφωνούν στη χρήση μια συνάρτησης κατακερματισμού από μια οικογένεια συναρτήσεων και ο αποστολέας κατακερματίζει το μήνυμα, το οποίο στη συνέχεια κρυπτογραφείται. Κατακερματισμός είναι η διαδικασία κατά την οποία το μήνυμα δίνεται σαν είσοδος στην συνάρτηση hash (κατακερματισμού) και παράγει ως αποτέλεσμα μια μοναδική και τυχαία σειρά χαρακτήρων που αντιπροσωπεύει το αρχικό μήνυμα. Για τον κατακερματισμό, το μήνυμα πολλαπλασιάζεται με τη χρήση ενός Toeplitz matrix έναν πίνακα στο οποίο όλα τα στοιχεία μιας διαγώνιου είναι ίδια και έχει πάρει το όνομα του από τον Γερμανό μαθηματικό που μελέτησε την οικογένεια αυτών των Matrix τον Otto Toeplitz.

T_0		K_{-1}	K_0	K_1	…	K_{L-2}	m_0
T1	=	K_{-2}	K_{-1}	K_0	…	K_{L-3}	m_1
…		⋮	⋮	⋮	⋱	⋮	…
T_{w-1}		K_{-w}	K_{-w-1}	K_{-w-2}	…	K_{L-1-w}	m_{L-1}

Σχήμα 8 - Toeplitz matrix defined by an ε-biased sequence

Ο πολλαπλασιασμός με το πίνακα Toeplitz επιτυγχάνεται με τη χρήση του καταχωρητή ολίσθησης (R) και του συσσωρευτή (A) ως εξής:

$$R_0 = [k_{-w}, \dots, k_{-1}], R_1 = [k_{-w+1}, \dots, k_0], \dots, R_{L-1} = [k_{L-1-w}, \dots, k_{L-2}]$$

Κάθε σειρά R_i οπου i ο αριθμός της σειράς πολλαπλασιάζεται αντίστοιχα με τη τιμή m_i, m_0 και m_1 έως m_{L-1} είναι οι παράγοντες που χρησιμοποιούνται για τον πολλαπλασιασμό. Επιπλέον το R_i είναι R_{i-1} μετακινημένο μια θέση αριστερά. Αυτό επιτυγχάνεται με τη χρήση ενός shift register (καταχωρητής ολίσθησης) με αρχική κατάσταση $R_0 = [k_{-w}, \dots, k_{-1}]$, και συσσωρετυή A όπου ισχύει

$$A \leftarrow A \oplus R_i m_i$$

Για παράδειγμα, έστω ότι το bit m_i του μηνύματος είναι 1, ανανεώνεται η κατάσταση του συσσωρευτή προσθέτοντας του το περιεχόμενο του καταχωρητή. Διαφορετικά αν είναι 0 δεν γίνεται τίποτα. Στη συνέχεια, γίνεται ολίσθηση προς τα αριστερά στο επόμενο bit στον καταχωρητή. Ένα keystream, το οποίο αποτελείται από 64-bit, χρησιμοποιείται για τη κρυπτογράφηση μια φορά με ένα τυχαίο κλειδί και ονομάζεται “one time pad”.

Το one time pad προστίθεται στο συσσωρευτή για την κρυπτογράφηση του hash. Αντί να πάρει το keystream block από το τέλος του keystream, το block εξάγεται κατά τη φάση της αρχικοποίησης της κρυπτογράφησης και συνεπώς ο συσσωρευτής αρχικοποιείται με τα bits του keystream block. Αφού αρχικοποιηθεί ο συσσωρευτής μπορούν να πραγματοποιηθούν επιπλέον ενέργειες για την κρυπτογράφηση του hash. Η πρόσθεση του keystream block στον συσσωρευτή κατά την αρχικοποίηση έχει ως αποτέλεσμα την εκκίνηση της κρυπτογράφησης με τυχαία και μοναδικά bits, αυξάνοντας την δύναμη της. Με τη χρήση συγκεκριμένου τύπου ακολουθίας για να οριστεί ο Toeplitz matrix, μπορεί να έχει σημαντική συνεισφορά στην ασφάλεια και την αρχικοποίηση του MAC. Στον Grain-128AEADv2, χρησιμοποιείται η ακολουθία ε-biased sequence η οποία εξάγεται από την συνάρτηση pre-output. Ορίζοντας έτσι Toeplitz matrix εξασφαλίζονται κάποιες ιδιότητες ασφαλείας του MAC, συγκεκριμένα η πιθανότητα αντικατάστασης (substitution probability) του MAC PS $\leq 2^{-w} + 2\varepsilon$ όπου w το μέγεθος του MAC. Αυτή η ανισότητα δίνει το μέγιστο όριο της πιθανότητας μη εγκεκριμένων παρεμβάσεων και έτσι παρουσιάζεται το επίπεδο ασφαλείας του MAC. Επιπλέον, η χρήση της ε-biased από την pre-output απλοποιεί την υλοποίηση του MAC σε υλικό, βελτιώνοντας την πρακτικότητα και την απόδοση του μηχανισμού ταυτοποίησης.

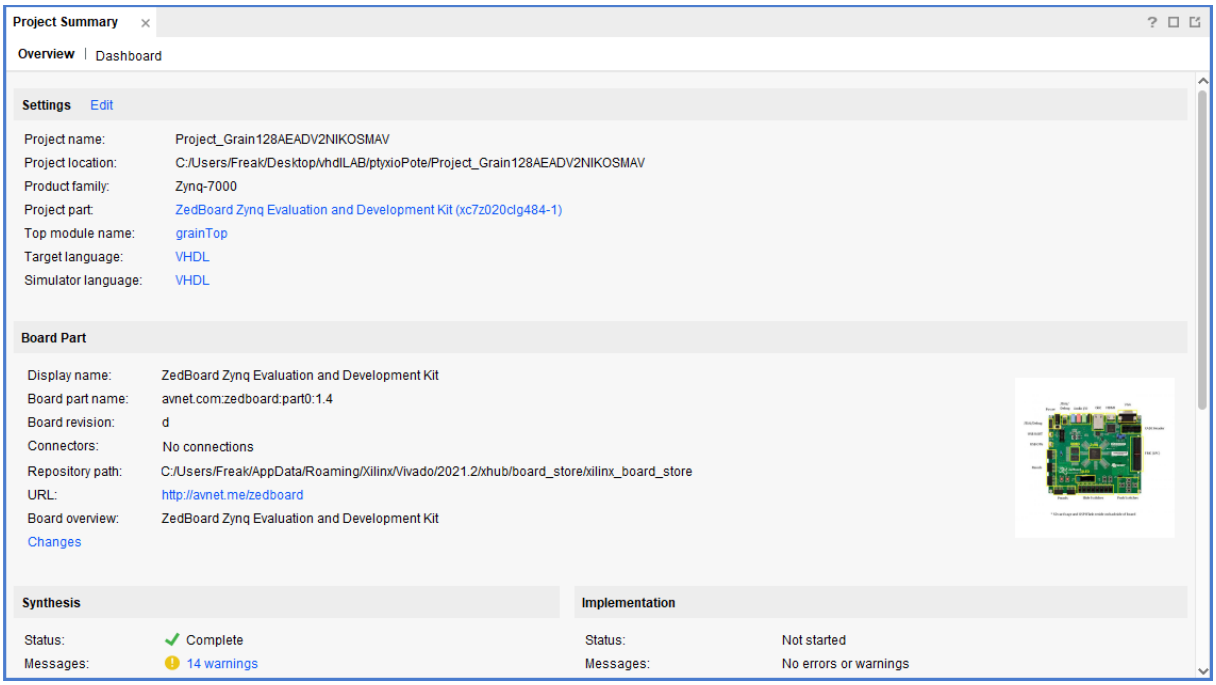
Υπάρχουν και άλλες πιθανές προσεγγίσεις με τις οποίες θα γινόταν η υλοποίηση πιο απλή ως προς το υλικό ή θα κατάφερνε να αυξήσει τη ταχύτητα ωστόσο, με τον προτεινόμενο τρόπο είναι δυνατή η συσχέτιση της ασφάλειας του MAC με την ασφάλεια του αλγορίθμου.

Κεφάλαιο 3

3.1 Σύνθεση αλγορίθμου Grain-128AEADv2

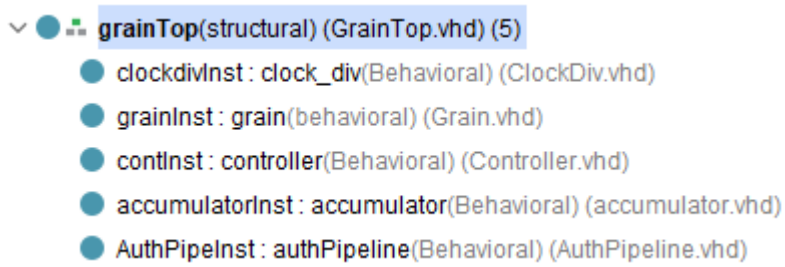
Στο κεφάλαιο αυτό παρουσιάζεται η σύνθεση του Grain-128AEADv2 και η προσομοίωση του με χρήση test-bench καθώς με χρήση του προγράμματος VIVADO 2021.2 και τα αποτελέσματα αυτών με στιγμιότυπα οθόνης και σχολιασμό τους.

Έχοντας δημιουργήσει το project στο περιβάλλον του VIVADO, τα χαρακτηριστικά του οποίου φαίνονται στο σχήμα 10, μπορεί να πραγματοποιηθεί η σύνθεση του συστήματος



Σχήμα 9 - Project overview του Grain-128AEADv2 στο VIVADO 2021.2

Ο Grain-128AEADv2 αποτελείται από 5 components τα οποία φαίνονται στο σχήμα 11, και ο κώδικας τους θα παρουσιαστεί στο τέλος της πτυχιακής. Αυτά τα components είναι απαραίτητα για τη λειτουργία του Grain-128AEADv2 και συνδυάζονται στο Top component το grainTop.



Σχήμα 10 – Components του Grain-128AEADv2

Αφού τρέξει η σύνθεση, η διαδικασία κατά την οποία ο κώδικας μετατρέπεται σε κύκλωμα το VIVADO παράγει μια αναφορά στην οποία μεταξύ άλλων δηλώνει τον αριθμό των σφαλμάτων εάν υπάρχουν, τη διάρκεια και τις διαδικασίες που ακολουθήθηκαν.

Αρχικά, παρουσιάζεται η σύνθεση των components του συστήματος με τίτλο Detailed RTL Component Info,

---

Detailed RTL Component Info :
+---Adders :
2 Input 1 Bit Adders := 1
+---XORs :
2 Input 32 Bit XORs := 1
6 Input 1 Bit XORs := 64
3 Input 1 Bit XORs := 64
16 Input 1 Bit XORs := 64
13 Input 1 Bit XORs := 64
2 Input 1 Bit XORs := 64
33 Input 1 Bit XORs := 64
+---Registers :
128 Bit Registers := 2
64 Bit Registers := 3
32 Bit Registers := 1

```
                4 Bit    Registers := 1
                1 Bit    Registers := 2
+---Muxes :
   2 Input   64 Bit       Muxes := 3
   2 Input   32 Bit       Muxes := 4
   2 Input    1 Bit       Muxes := 128
```

Finished RTL Component Statistics

Ακολουθεί η τελική αναφορά αφού έχει τελειώσει η διαδικασία της σύνθεσης. Σε αυτή συμπεριλαμβάνεται η διάρκεια, τα αρχεία που μεταγλωττίστηκαν, καθώς και η μνήμη που χρησιμοποιήθηκε.

Start Writing Synthesis Report

Report BlackBoxes:

```
+-+--------------+----------+
| |BlackBox name |Instances |
+-+--------------+----------+
+-+--------------+----------+
```

Report Cell Usage:

```
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |BUFG |     1|
|2     |LUT1 |     1|
|3     |LUT2 |   212|
|4     |LUT3 |   238|
|5     |LUT4 |  1017|
|6     |LUT5 |   428|
```

```
|7     |LUT6 |   636|
|8     |FDCE |   489|
|9     |FDPE |     1|
|10    |IBUF |   165|
|11    |OBUF |    97|
+------+-----+------+


Report Instance Areas:
+------+-----------------+-------------+------+
|      |Instance         |Module       |Cells |
+------+-----------------+-------------+------+
|1     |top              |             |  3285|
|2     |  AuthPipeInst   |authPipeline |   587|
|3     |  accumulatorInst|accumulator  |   918|
|4     |  clockdivInst   |clock_div    |     2|
|5     |  contInst       |controller   |   263|
|6     |  grainInst      |grain        |  1252|
+------+-----------------+-------------+------+
```

Finished Writing Synthesis Report : Time (s): cpu = 00:02:18 ; elapsed = 00:02:16 . Memory (MB): peak = 1266.953 ; gain = 12.738

Synthesis finished with 0 errors, 0 critical warnings and 14 warnings.

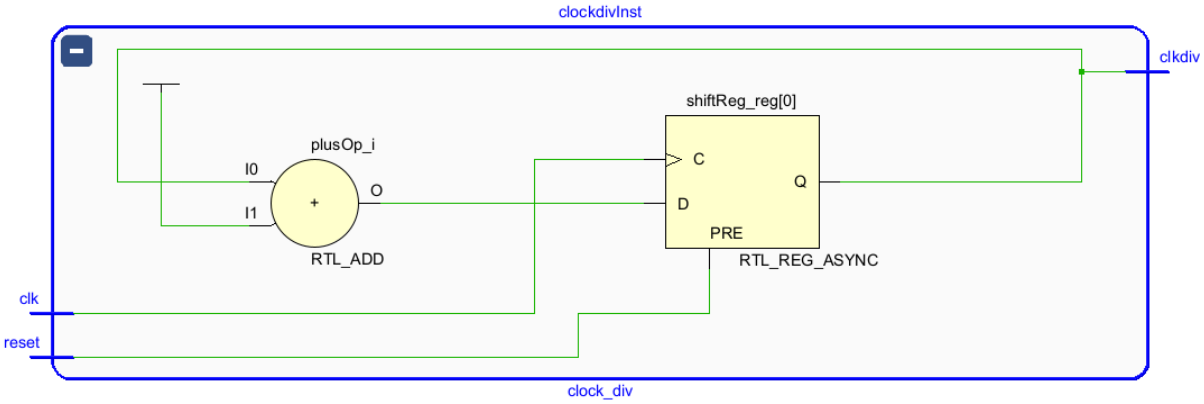
Synthesis Optimization Runtime : Time (s): cpu = 00:02:18 ; elapsed = 00:02:16 . Memory (MB): peak = 1266.953 ; gain = 12.738

Synthesis Optimization Complete : Time (s): cpu = 00:02:18 ; elapsed = 00:02:16 . Memory (MB): peak = 1266.953 ; gain = 12.738

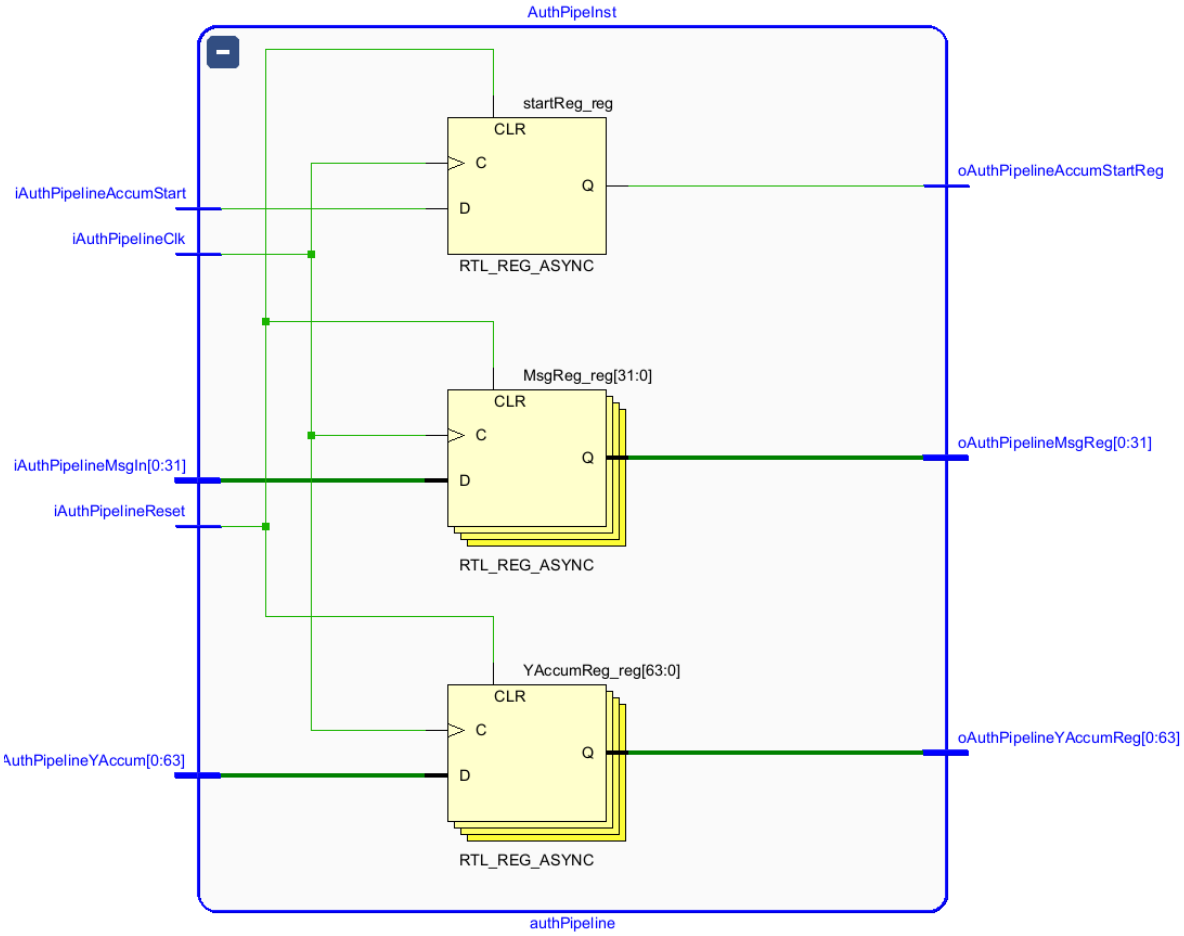
3.2 Παρουσίαση κυκλώματος Grain-128AEADv2

Σε αυτό το κεφάλαιο παρουσιάζεται το κύκλωμα (schematic) του Grain-128AEADv2 μέσα από στιγμιότυπα οθόνης. Πιο αναλυτικά, φαίνεται το κύκλωμα των components, ο τρόπος που όλα συνδέονται μεταξύ τους και οι είσοδοι και έξοδοι του κυκλώματος. Το κύκλωμα

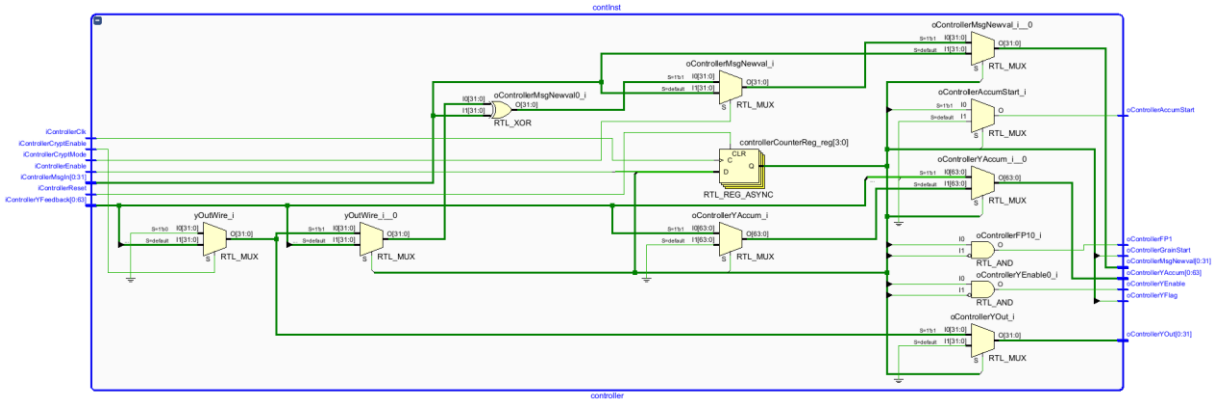
έχει δημιουργηθεί από το VIVADO 2021.2 βάσει της περιγραφής που δόθηκε από τα αρχεία κώδικα τα οποία μεταγλωττίστηκαν κατά τη σύνθεση.



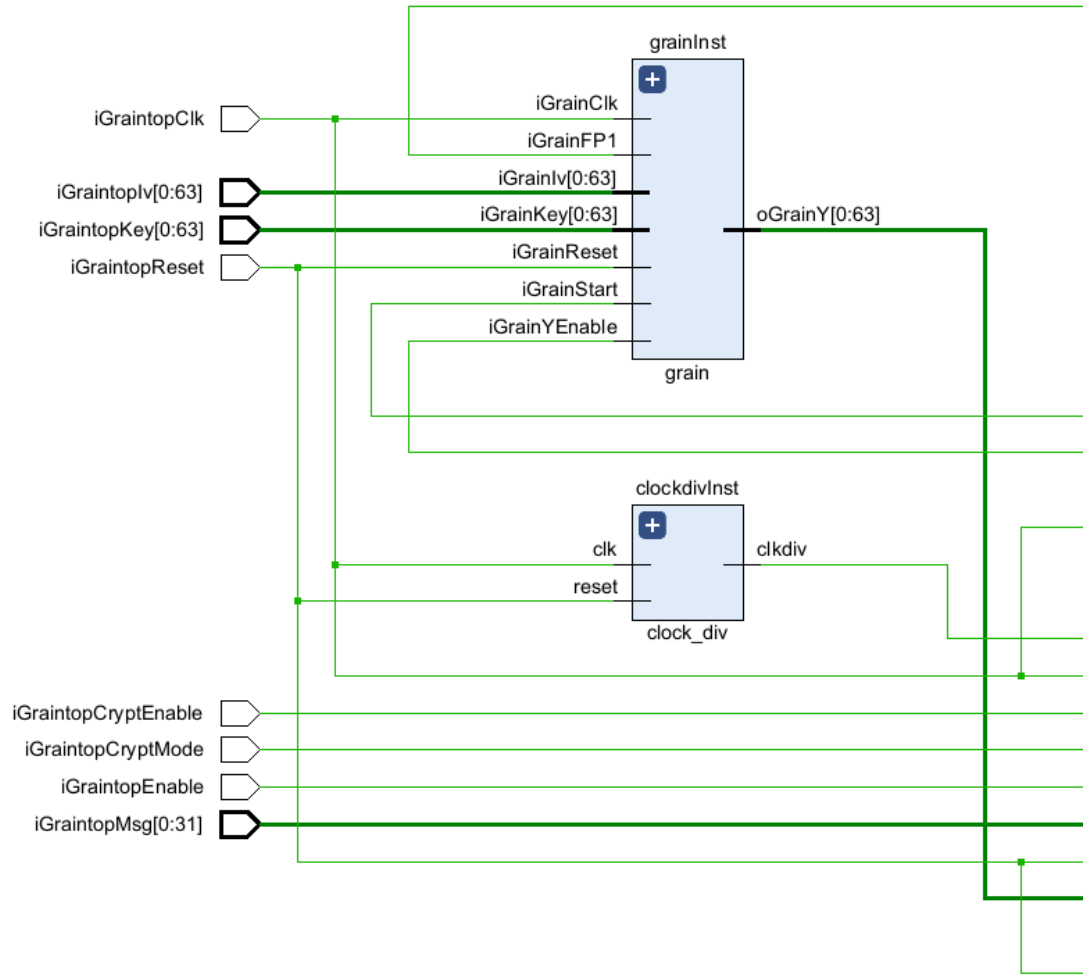
Σχήμα 11 - Schematic overview του ClockDiv



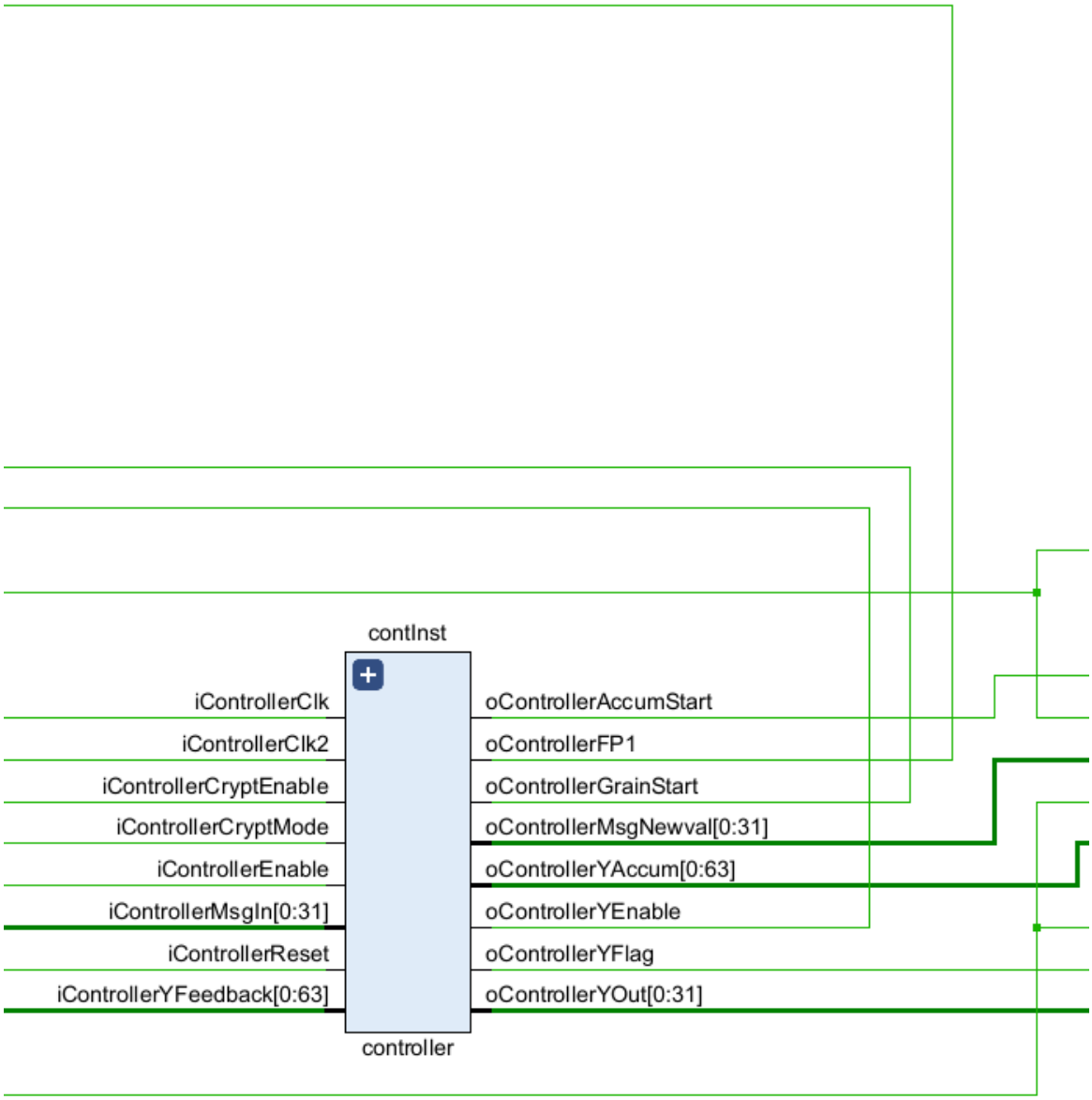
Σχήμα 12 - Schematic overview του AuthPipeline



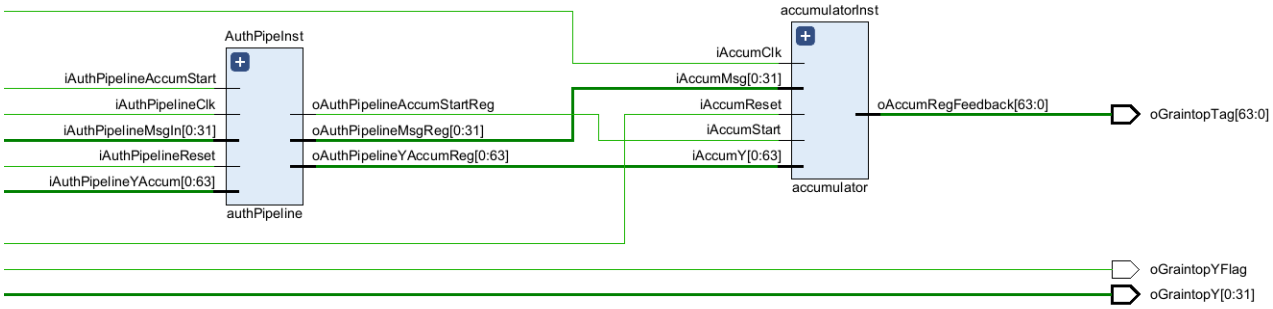
Σχήμα 13 - Schematic overview του Controller



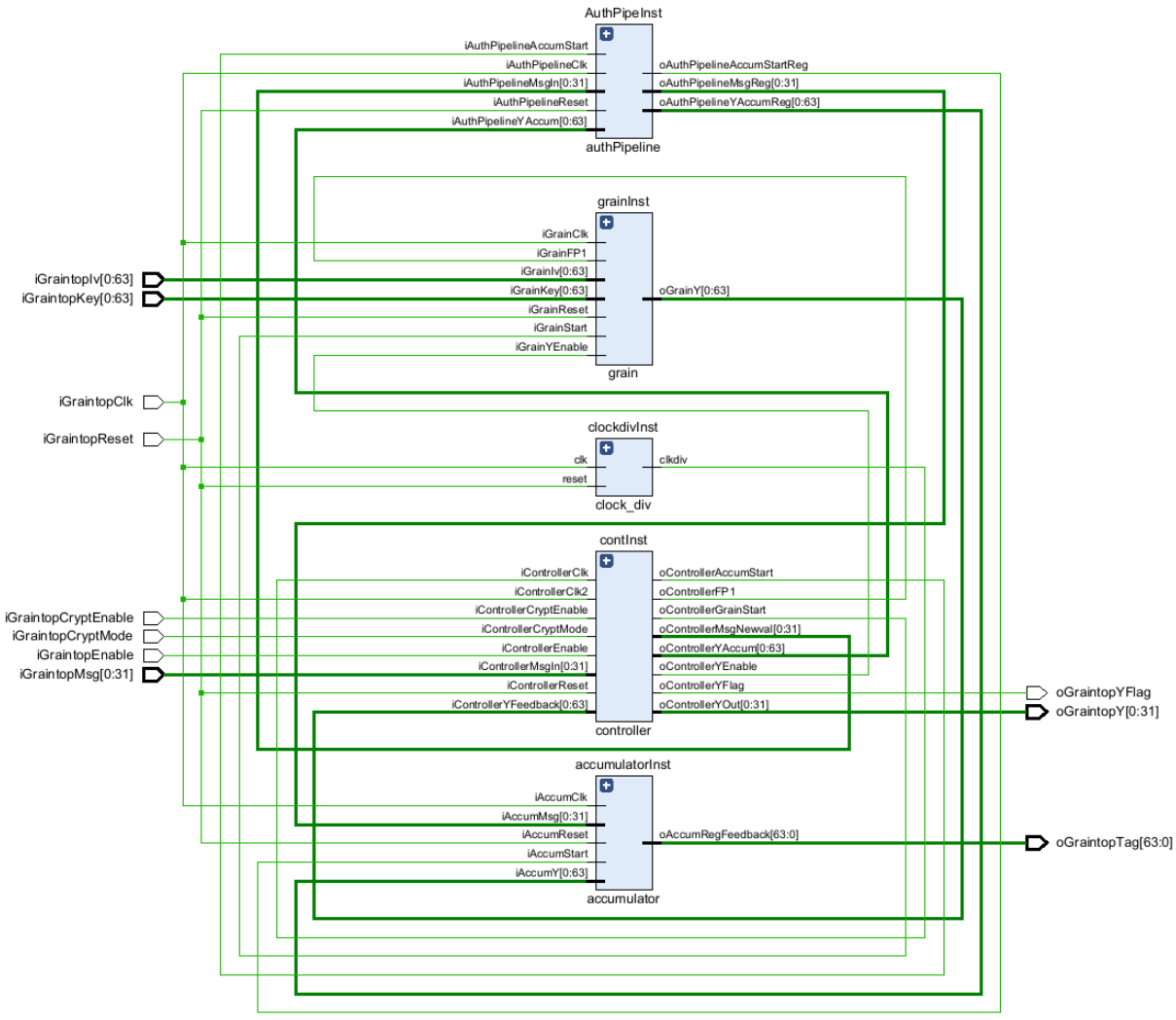
Σχήμα 14 - Schematic overview του GrainTop - Part 1



Σχήμα 15 - Schematic overview του GrainTop - Part 2



Σχήμα 16 - Schematic overview του GrainTop - Part 3



Σχήμα 17 - Schematic overview του GrainTop Full size

3.3 Λειτουργική προσομοίωση αλγορίθμου Grain-128AEADv2

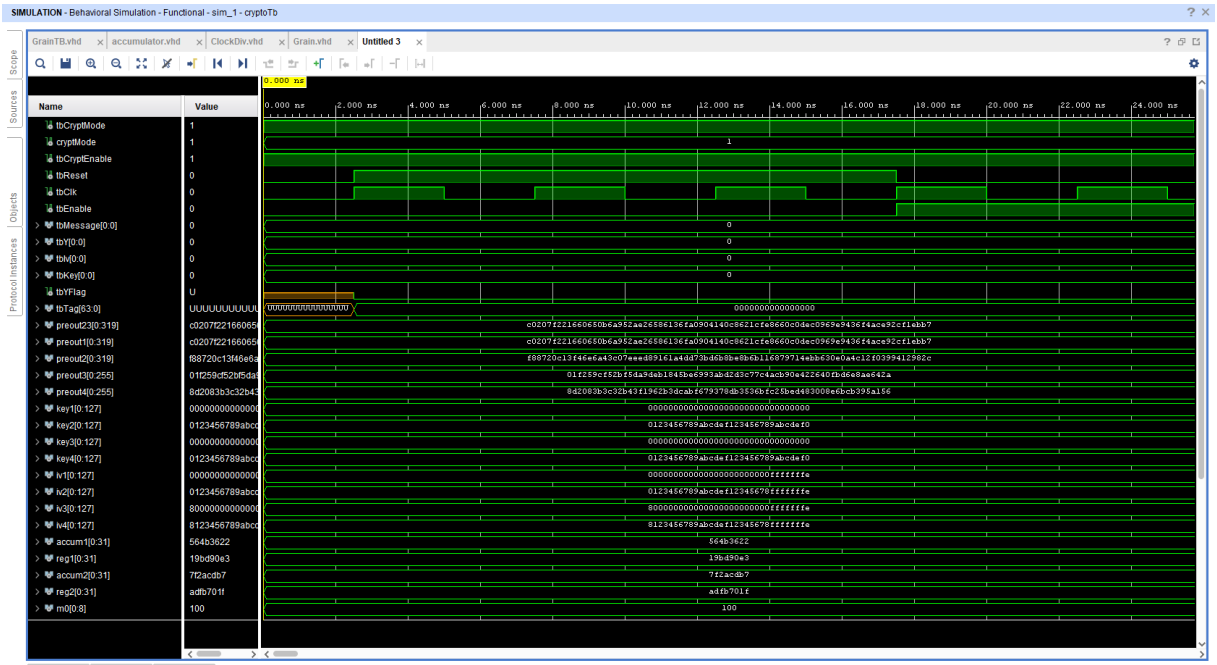
Στο κεφάλαιο αυτό θα παρουσιαστεί η λειτουργική προσομοίωση του κυκλώματος, η οποία επιτυγχάνεται με τη χρήση ενός testbench. Testbench είναι ένα αρχείο κώδικα με το οποίο επιβεβαιώνεται η σωστή λειτουργία ενός κυκλώματος.

Το testbench που ελέγχει τη λειτουργία του κυκλώματος ονομάζεται GrainTB ο κώδικας του υπάρχει στη συνέχεια και λειτουργεί με τον εξής τρόπο. Σε πρώτη φάση αρχικοποιούνται τα σήματα, οι σταθερές, δημιουργείται το portmap και ένα generic map(γενικός χάρτης) σημάτων του GrainTop που είναι και το σύστημα το οποίο ελέγχεται. Το generic map αντιστοιχεί τιμές με μεταβλητές όπως n, div2, g, yt, ctype, clkdivnum, and ia, και περιέχει δικά του σήματα όπως σήμα ρολογιού tbClk και σήματα ελέγχου όπως, tbReset, tbEnable, tbCryptEnable, tbCryptMode. Με τα σήματα αυτά ορίζει τη λειτουργία κατά τη προσομοίωση. Μέσα στο "process block" υπάρχει η λούπα προσομοίωσης όπου δίνονται διάφορες τιμές στις μεταβλητές t, m, f και k. Οι μεταβλητές αυτές ελέγχουν τη ροή της προσομοίωσης και δημιουργούν το αρχείο stimulus μέσα στο οποίο περιέχονται όλες οι απαραίτητες τιμές. Στη συγκεκριμένη περίπτωση, περιγράφονται διαφορετικά σενάρια τα οποία εκτελούνται ανάλογα με τη τιμή των προ αναφερθέντων μεταβλητών. Το testbench παρέχει το stimuli για το component το οποίο ελέγχεται (Design Under Test – DUT) και με αυτόν τον τρόπο γίνεται ανάθεση τιμών στα εισερχόμενα σήματα του DUT. Τα εισερχόμενα σήματα περιλαμβάνουν το μήνυμα που θα επεξεργαστεί σήματα ελέγχου και τις εισόδους που απαιτούνται. Κατόπιν, εκτελούνται οι εντολές ελέγχου, οι οποίες περιλαμβάνουν λογικούς ελέγχους, μέτρηση σημάτων εξόδου και επαλήθευση των αποτελεσμάτων. Με την αξιοποίηση διαφορετικών σεναρίων επιτυγχάνεται ο εξονυχιστικός έλεγχος της σωστής λειτουργίας του συστήματος και η ανίχνευση πιθανών σφαλμάτων. Τα αποτελέσματα της προσομοίωσης καταγράφονται και αξιολογούνται μέσω αναφορών και γραφικών παραστάσεων, καθώς και συγκρίνοντας τα αναμενόμενα αποτελέσματα με τα πραγματικά.

Κατά την προσομοίωση η διαδικασία που ακολουθείται είναι η εξής:

Ξεκινάει μια λούπα τεσσάρων επαναλήψεων για t από 0 έως 3(για τις 4 περιπτώσεις) και μέσα σε αυτή άλλη λούπα τριών επαναλήψεων για m από 0 έως 2. Αρχικοποιούνται οι μεταβλητές «MsgIndex» και «MsgFinish» με τιμή 0 και στη συνέχεια υπάρχει αναμονή για 3 περιόδους, όπου η περίοδος είναι 5ns. Αρχικοποίηση των σημάτων tbReset και tbEnable, επανάληψη 2 φορές για τη μεταβλητή f από 0 έως 1, και στη συνέχεια επανάληψη (128/n) φορές για τη μεταβλητή k από 0 έως (128/n) – 1. Ανάλογα με την τιμή της μεταβλητής t, ορίζονται τα σήματα «tbKey» και «tbIv» από τις μεταβλητές «key1», «iv1», «key2», «iv2», «key3», «iv3», «key4» και «iv4». Εάν η μεταβλητή f είναι ίση με 1, το σήμα «tbMessage» αρχικοποιείται με μηδενικά και εάν η μεταβλητή k είναι ίση με (128/(2*n)), η πρώτη θέση του σήματος «tbMessage» παίρνει τιμή 1. Μετά το πέρασμα μιας περιδόδου ελέγχεται η μεταβλητή f όπου αν έχει τιμή 0 τότε τα σήματα «tbKey» και «tbIv» παίρνουν τη τιμή 0 και αναμονή έως το σήμα «tbYFlag» πάρει τη τιμή 1, διαφορετικά αν η μεταβλητή f έχει τιμή 1 τότε τα σήματα «tbKey» και «tbIv» παίρνουν τη τιμή 0 και «tbYFlag» δεν δέχεται κάποια αλλαγή. Συνεχίζοντας ορίζεται το σήμα «tbCryptEnable» σε 1 και ανάλογα με τη τιμή της μεταβλητής m προχωράει και ο αλγόριθμος όπου αν m=0 και «MsgFinish» επίσης 0,

αρχικοποιείται το σήμα «tbMessage» με μηδενικά. Ακολουθεί επανάληψη για το g από 0 έως m0'LENGTH – 1(μήκος μυνήματος -1). Μεσα στην επανάληψη αυτή ορίζεται το σύνολο των σημάτων «tbMessage» ανάλογα με τη τιμή του g και του ndiv2. Με την εντολή, «tbMessage((g mod (ndiv2))) <= m0(g);» τοποθετεί την τιμή του m0(g) στη θέση «(g mod (ndiv2))» του σήματος tbMessage. Η τιμή του m0(g) αποστέλλεται στο tbMessage με το πιο σημαντικό ψηφίο να αποστέλλεται πρώτο. Στη συνέχεια, υπάρχει η συνθήκη «if ((g mod (ndiv2)) = ndiv2 - 1) then», που ελέγχει εάν η τιμή της g είναι ίση με ndiv2 - 1. Αν αυτή η συνθήκη ισχύει, εκτελούνται οι παρακάτω ενέργειες, tbMessage παίρνει μηδενική τιμή για να επαναφέρει την τιμή του για τον επόμενο γύρο του μηνύματος. Εάν η τιμή της n είναι 1, υπάρχει αναμονή μιας περιόδου. Τέλος, υπάρχει ένας έλεγχος για το εάν η τιμή της g είναι ίση με m0'LENGTH - 1. Αν αυτή η συνθήκη ισχύει, τοτε το σήμα tbMessage παίρνει τιμή 0 για να επαναφέρει την τιμή του για τον επόμενο γύρο του μηνύματος. Τέλος, η μεταβλητή MsgFinish λαμβάνει τιμή 1, προκαλώντας τη λήξη της κρυπτογράφησης/αποκρυπτογράφησης. Το tbMessage επαναφέρεται σε μηδενικές τιμές. Η δεύτερη περίπτωση αφορά το m=1. Εάν το MsgFinish είναι 0, ο κώδικας εκτελεί τις ακόλουθες ενέργειες: Η μεταβλητή tbMessage επαναφέρεται σε μηδενικές τιμές. Εκτελείται ένας βρόχος για g από 0 έως m1'LENGTH - 1. Εντός του βρόχου, αναθέτεται η τιμή του m1(g) στο κατάλληλο ψηφίο του tbMessage. Εάν (g mod (ndiv2)) είναι ίσο με (ndiv2) - 1, υπάρχει καθυστέρηση μιας περιόδου και το tbMessage επαναφέρεται σε μηδενικές τιμές. Εάν το n είναι ίσο με 1, εισάγεται μια επιπλέον καθυστέρηση μιας περιόδου. Εάν το g είναι ίσο με m1'LENGTH - 1, πραγματοποιούνται οι ακόλουθες ενέργειες: καθυστέρηση μιας περιόδου, και επαναφορά του tbMessage μηδενικές τιμές. Μετά τον βρόχο, η τιμή του MsgFinish ορίζεται σε 1. Ο κώδικας ορίζει το tbCryptEnable σε 0. Το tbMessage επαναφέρεται σε μηδενικές τιμές. Αντίστοιχα εκτελούνται οι εντολές για την 3η περίπτωση όπου m=2.



Σχήμα 18 - Behavioral simulation of Grain using GrainTB.vhd

```
--------------------------------------------------------------------------------------------
--test1

key1 <= X"00000000000000000000000000000000";
iv1  <= X"00000000000000000000000ffffffe";
iv1 <= X"00000000000000000000000100000000";
preout1 <= X"c0207f221660650b6a952ae26586136fa0904140c8621cfe8660c0dec0969e9436f4ace92cf1ebb7";

---test2
key2 <= X"0123456789abcdef123456789abcdef0";
key2 <= (others=>'0');
iv2      <= X"0123456789abcdef12345678fffffffe"; -- 80c4a2e691d5b3f7482c6a1e

preout23 <= X"c0207f221660650b6a952ae26586136fa0904140c8621cfe8660c0dec0969e9436f4ace92cf1ebb7";
preout2  <= X"f88720c13f46e6a43c07eeed89161a4dd73bd6b8be8b6b116879714ebb630e0a4c12f0399412982c";
----test3
key3    <= (others => '0');
iv3     <= X"80000000000000000000000ffffffe";
accum1  <= X"564b3622"; --new add
reg1    <= X"19bd90e3"; --new add
preout3 <= X"01f259cf52bf5da9deb1845be6993abd2d3c77c4acb90e422640fbd6e8ae642a";


----test4
key4    <= X"0123456789abcdef123456789abcdef0";
iv4     <= X"8123456789abcdef12345678fffffffe";
accum2  <= X"7f2acdb7"; --new add
reg2    <= X"adfb701f"; --new add
preout4 <= X"8d2083b3c32b43f1962b3dcabf679378db3536bfc25bed483008e6bcb395a156";

------5 different Message
m0 <= "100000000";
m1 <= "01";
m2 <= "11";
m3 <= "000100100011010000001";

m4 <= "00010010001101000101011001111000100111101l";
```

Σχήμα 19 - Test vectors of Grain-128AEADv2

Κεφάλαιο 4

Πλεονεκτήματα και περιορισμοί του Grain-128AEADv2

4.1 Καταλληλόλητα του Grain-128AEADv2 για χρήση σε συστήματα IOT

Έχουν παρατηρηθεί τα πλεονεκτήματα της οικογένειας Grain σε συστήματα IOT και ενσωματωμένα συστήματα στο χώρο της βιομηχανίας. Τα τελευταία χρόνια τα RFID συστήματα έχουν εδραιωθεί σε διάφορους τομείς της βιομηχανίας και υπάρχει μεγάλη ανάγκη για την προστασία των συστημάτων αυτών. Ο διαγωνισμός eSTREAM, έχει αναδείξει ευρέως την οικογένεια του Grain για την απόδοση τους και την ασφάλεια που παρέχουν. Είναι, πλέον, εμφανής η συμβολή της τεχνολογίας σε όλους τους τομείς της αυτοκινητοβιομηχανίας, τόσο στην εφαρμογή νέων τεχνολογιών κατά τη σχεδίαση των αυτοκινήτων με χρήση λογισμικών όπως το Autosar (AUTomotive Open System ARchitecture), όσο και στο τελικό προϊόν, το ίδιο το αυτοκίνητο όπου λειτουργίες όπως η PKES (Passive Keyless Entry and Start), αυτόματη είσοδος και εκκίνηση του οχήματος, απλά πλησιάζοντας το είναι πλέον εφικτές. Η μονάδα ελέγχου (ECU, Electronic Control Unit) εντοπίζει το κλειδί, επικοινωνεί μαζί του, και πραγματοποιεί τις απαραίτητες διεργασίες. Όλα αυτά γίνονται σε πραγματικό χρόνο και η ασφαλής επικοινωνία είναι υψίστης σημασίας. Από το 2010 και μετά, έχουν πραγματοποιηθεί επιθέσεις [17] στα συστήματα PKES. Για να εξασφαλιστεί η ασφάλεια απέναντι σε αυτές τις επιθέσεις είναι απαραίτητη η χρήση αλγορίθμων κρυπτογραφίας. Αναμένεται ότι ένα κρυπτογραφικό πρωτότυπο θα υλοποιηθεί σε υλικό στο κλειδί (RFID ετικέτα) και σε ενσωματωμένο λογισμικό σε κάποια μονάδα ελέγχου του οχήματος, πράγμα που σημαίνει ότι το κρυπτογραφικό πρωτότυπο πρέπει να πληροί τόσο τις απαιτήσεις υλικού όσο και τις απαιτήσεις ενσωματωμένου λογισμικού. Ο grain-128AEADv2 μπορεί να αποτελέσει επιλογή για την διασφάλιση της επικοινωνίας αυτής λόγω της ανταγωνιστικής του απόδοσης σε υλικό, καθώς και της πολύ καλής του απόδοσης σε ενσωματωμένο λογισμικό. Ένα ακόμα πλεονέκτημα του Grain-128AEADv2 για εφαρμογές IoT μπορεί να παρατηρηθεί στη χρήση μάσκας. Επιπλέον ο Grain-128AEADv2 είναι στενά βασισμένος στο Grain-128a, ο οποίος έχει αναλυθεί εκτενώς από τρίτους. Επίσης, οι προκάτοχοί του, Grain v1 και Grain128, έχουν υποβληθεί σε λεπτομερή ανάλυση από την πρώτη εμφάνιση του Grain το 2005. Ο Grain128AEADv2 δεν αποθηκεύει το κλειδί σε μη αποσβέσιμη μνήμη. Αυτό έχει τον περιορισμό ότι το μέγεθος της εσωτερικής κατάστασης είναι μεγαλύτερο, αλλά ταυτόχρονα έχει το πλεονέκτημα ότι το κλειδί μπορεί να ενημερωθεί στη συσκευή, καθιστώντας το χρήσιμο για ευρύτερη γκάμα περιπτώσεων. Ο Grain-128AEADv2, όπως και οι άλλοι κρυπτοαλγόριθμοι της οικογένειας, έχουν σχεδιαστεί με σκοπό να αυξήσουν την ταχύτητα χωρίς την προσθήκη επιπλέον υλικού. Αυτό παρέχει μια ευρεία γκάμα περιπτώσεων χρήσης, όπως καταστάσεις όπου η χρήση ελάχιστου υλικού είναι η προτεραιότητα, αλλά και περιπτώσεις όπου η ταχύτητα είναι πιο σημαντική.

Κεφάλαιο 5

Συμπεράσματα

Σε αυτό το κεφάλαιο θα γίνει αναφορά στα συμπεράσματα που προκύπτουν με την ολοκλήρωση της παρούσας πτυχιακής εργασίας, σκοπός της οποίας ήταν η μελέτη και ανάλυση του αλγορίθμου κρυπτογραφίας Grain-128AEADv2, ο οποίος προτάθηκε στο διαγωνισμό του NIST (National Institute of Standards and Technology). Μετά από σύντομη περιγραφή του περιεχομένου της παρούσας πτυχιακής, στο 2ο κεφάλαιο παρουσιάστηκε ο Grain-128AEADv2 και αναλύθηκε ο τρόπος λειτουργίας του, τα blocks που τον απαρτίζουν, καθώς και μια σύντομη αναδρομή στην πορεία που έχει ακολουθήσει η οικογένεια Grain μέχρι τον Grain-128AEADv2. Έπειτα, ακολούθησε η υλοποίηση του κώδικα και η διαδικασία της σύνθεσης με τελικό σκοπό την προσομοίωση λειτουργίας μέσω του VIVADO 2021.2. Όλα τα αρχεία κώδικα που χρησιμοποιήθηκαν παρουσιάζονται στο ΠΑΡΑΡΤΗΜΑ Α.

Ο Grain-128AEADv2 ανήκει σε μια μεγάλη οικογένεια αλγορίθμων ροής, η οποία είναι καλά εδραιωμένη στο χώρο της κρυπτογραφίας. Αποτέλεσμα αυτής της συνθήκης αποτελεί το γεγονός ότι από το 2005 έως και σήμερα, πολλοί ερευνητές έχουν μελετήσει την ανθεκτικότητα, την ταχύτητα και γενικότερα τις δυνατότητες της οικογένειας αυτής, η οποία φαίνεται πως έχει θέσει γερά θεμέλια και σε κάποιες περιπτώσεις αποτελεί και το standard στις επικοινωνίες. Είναι γρήγορος αλγόριθμος χωρίς σημαντικές ανάγκες υλικού και η structural δομή του επιτρέπει στον εκάστοτε χρήστη να τον μεταβάλλει ανάλογα με τις ανάγκες του. Ο τρόπος με τον οποίο είναι γραμμένος επιτρέπει στον αναλυτή που θα τον μελετήσει να κατανοήσει με σχετική ευκολία τις λειτουργίες του.

Η ανάλυση του τρόπου λειτουργίας του Grain-128AEADv2 είχε ως αποτέλεσμα την εξοικείωση με τη γλώσσα VHDL και τους κανόνες της, αλλά και με το VIVADO design suite, πρόγραμμα το οποίο χρησιμοποιήθηκε για τη σύνθεση και ανάλυση του αλγορίθμου. Η χρήση είναι αρκετά απλή ακόμα και για κάποιον με ελάχιστη εμπειρία, γεγονός το οποίο έκανε τη δημιουργία του project αρκετά εύκολη. Με τη χρήση του VIVADO ο εντοπισμός σφαλμάτων γινόταν σε πραγματικό χρόνο τόσο για λογικά λάθη όσο και για συντακτικά. Τέλος, λόγω της λειτουργίας της προσομοίωσης κάθε αλλαγή στο κώδικα, μικρή ή μεγάλη, ήταν πολύ εύκολο να παρατηρηθεί, αφού το πρόγραμμα παρέχει τη δυνατότητα της step by step simulation όπου όλη η διαδικασία γίνεται βήμα προς βήμα. Βάσει των παραπάνω, εργαλεία σαν αυτό είναι απαραίτητα και έχουν σημαντικό ρόλο στο σχεδιασμό, τον έλεγχο και την ανάλυση συστημάτων.

Βιβλιογραφία

Βιβλία:

- Σχεδιασμός κυκλωμάτων με τη VHDL, Pedroni A. Volnei
- Λογική σχεδίαση ψηφιακών συστημάτων, ΠΑΠΑΟΔΥΣΣΕΥΣ Κ., ΕΞΑΡΧΟΣ ΜΙΧΑΗΛ, ΑΡΑΜΠΑΤΖΗΣ ΔΗΜΗΤΡΙΟΣ, ΓΙΑΝΝΟΠΟΥΛΟΣ ΦΩΤΙΟΣ

Ιστότοποι:

- Grain-128AEAD-VHDL/src at master · Noxet/Grain-128AEAD-VHDL · location of the code at GitHub by Noxet A.K.A. Jonathan Sönnerup [3]
- https://en.wikipedia.org/wiki/Grain_(cipher) [4]
- https://blog.checkpoint.com/security/the-tipping-point-exploring-the-surge-in-iot-cyberattacks-plaguing-the-education-sector/ [5]
- https://www.forbes.com/sites/zakdoffman/2019/09/14/dangerous-cyberattacks-on-iot-devices-up-300-in-2019-now-rampant-report-claims/?sh=513b75a05892 [6]
- M. Hell and T. Johansson, "Algebraic attacks and decompositions of the grain family of stream ciphers," in Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), Vienna, Austria, September 2007, pp. 207-221. [7]
- Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic attacks and decomposition of boolean functions. In International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2004, pages 474–491. Springer, 2004. [8]
- "Grain: A Stream Cipher for Constrained Environments" by Martin Hell, Thomas Johansson , and Willi Meier International Journal of Wireless and Mobile Computing 2007 [9]
- "Grain-128: Providing Robust Security for Wireless Applications" by Martin Hell, Thomas Johansson, and Willi Meier [10]
- "Cryptanalysis of Grain" by Alex Biryukov, Adi Shamir, and David Wagner [11]
- "On the Security of Grain-128" by Eli Biham, Orr Dunkelman, Nathan Keller, and Adi Shamir [12]

- "Improved Cryptanalysis of Grain-128 using Time-Memory Tradeoff" by Hongjun Wu and Bart Preneel [13]
- Lightweight Cryptography | CSRC (csrc.nist.gov/Projects/lightweight-cryptography) [14]
- "Fast Correlation Attack Revisited Cryptanalysis on Full Grain-128a, Grain-128, and Grain-v1" by Yosuke Todo , Takanori Isobe , Willi Meier , Kazumaro Aoki , and Bin Zhang [15]
- "Recovering the key from the internal state of grain-128aead" by Donghoon Chang and Meltem Sonmez Turan [16]
- Itai Dinur, Tim G¨uneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. An experimentally verified attack on full Grain-128 using dedicated reconfigurable hardware. [17]
- Paper 2012/617 Security Analysis of an Open Car Immobilizer Protocol Stack Stefan Tillich and Marcin Wójcik [18]
- Cyberattacks On IOT Devices Surge 300% In 2019, ‘Measured In Billions’ https://www.forbes.com/sites/zakdoffman/2019/09/14/dangerous-cyberattacks-on-iot-devices-up-300-in-2019-now-rampant-report-claims/ [19]
- The Tipping Point: Exploring the Surge in IoT Cyberattacks Globally https://blog.checkpoint.com/security/the-tipping-point-exploring-the-surge-in-iot-cyberattacks-plaguing-the-education-sector/ [20]

ΠΑΡΑΡΤΗΜΑ Α

Περιεχόμενα κώδικα VHDL

Αλγόριθμος Grain-128AEADv2

Το σύνολο του κώδικα υπάρχει στο ακόλουθο σύνδεσμο,

https://github.com/Grain-128AEAD/Grain-128AEAD-VHDL/tree/master by Noxet A.K.A. Jonathan Sönnerup

Αρχικά παρουσιάζεται το περιεχόμενο του αρχείου Grain.vhd,

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;

--------------------------------------------------------------------------------

entity grain is
    generic (
        n  : integer; -- Parallelization level
        g  : integer; -- Galois if 1.
        yt : integer -- Y transform / pipeline if 1
    );
    port (
        iGrainStart   : in std_logic; -- When 1 the cipher runs.
        iGrainFP1     : in std_logic; -- Active when doing FP1
        iGrainYEnable : in std_logic;
        iGrainIv      : in std_logic_vector(0 to n - 1);
        iGrainKey     : in std_logic_vector(0 to n - 1);
        oGrainY       : out std_logic_vector(0 to n - 1);
        iGrainReset   : in std_logic;
        iGrainClk     : in std_logic
    );
end grain;

--------------------------------------------------------------------------------

architecture behavioral of grain is
    --------------------------------------------------------------------------------
    -- Define the two registers: LFSR & NFSR (You can't use the same signal for input & output in hardware)
    signal grainLfsrReg              : std_logic_vector(0 to 127); -- Current value
    signal grainLfsrNext             : std_logic_vector(0 to 127); -- Next value
    signal grainNfsrReg              : std_logic_vector(0 to 127); -- Current value
    signal grainNfsrNext             : std_logic_vector(0 to 127); -- Next value
    signal enableReg, enableNext       : std_logic;
    signal enableReg2, enableNext2     : std_logic;

    signal PipelineReg, PipelineNext   : std_logic_vector(0 to n - 1); -- When using Y transform you need these
registers for pipelinning.
    signal PipelineReg2, PipelineNext2 : std_logic_vector(0 to n - 1); -- If they are unused they are removed
automatically in synthesis script at least in synposis.
```

```
    signal PipelineReg3, PipelineNext3 : std_logic_vector(0 to n - 1); -- If they are unused they are removed
automatically in synthesis script at least in synposis.
    -- (-32, 0, 0) (-16, 0, 0) (-32, 0, 32), (-32, 0, 16) (-32, 0, 15) (-31, 0, 15) (-31, 0, 16) (-16, 0, 16)
    constant tconst  : integer := 0; -- Just using this when trying to find a bug with the transformations.
    constant tconst2 : integer := 32; -- For Y transform
    constant tconst3 : integer := 0; -- For the first values in F and G (the shift value)
    ---------------------------------------------------------------------------

begin
    ---------------------------------------------------------------------------
    seq : process (iGrainClk, iGrainReset)
    begin
        if (iGrainReset = '1') then
            grainNfsrReg <= (others => '0');
            grainLfsrReg <= (others => '0');
            PipelineReg  <= (others => '0');
            PipelineReg2 <= (others => '0');
            PipelineReg3 <= (others => '0');
            enableReg    <= '0';
            enableReg2   <= '0';

        elsif (rising_edge(iGrainClk)) then
            grainNfsrReg <= grainNfsrNext;
            grainLfsrReg <= grainLfsrNext;
            PipelineReg  <= PipelineNext;
            PipelineReg2 <= PipelineNext2;
            PipelineReg3 <= PipelineNext3;
            enableReg    <= enableNext;
            enableReg2   <= enableNext2;
        end if;
    end process;

    ---------------------------------------------------------------------------
    comb : process (iGrainReset, grainLfsrReg, PipelineReg, PipelineReg2, PipelineReg2, enableReg,
enableReg2,
        grainNfsrReg, iGrainIv, iGrainKey, iGrainYEnable, iGrainstart, iGrainFP1)

        variable fWire  : std_logic_vector(0 to 127 + n + g * ( - n + 32)); -- The last values after 127 are the new
values generated for the NFSR and LFSR.
        variable gWire  : std_logic_vector(0 to 127 + n + g * ( - n + 32)); -- For galois we need a little more
variable space, max will be 32.

        variable yWire  : std_logic_vector(0 to n - 1);
        variable yWire2 : std_logic_vector(0 to n - 1);
        variable yWire3 : std_logic_vector(0 to n - 1); -- Allowing for 3 pipelines.
    begin
        fWire(0 to 127) := grainLfsrReg; -- We store the last values at
        gWire(0 to 127) := grainNfsrReg;
        grainLfsrNext <= grainLfsrReg(n to 127) & iGrainIv; -- Shift to the left.
        grainNfsrNext <= grainNfsrReg(n to 127) & iGrainKey; -- Default values. when loading

        enableNext    <= iGrainYEnable;
        enableNext2   <= enableReg;

        -- For loop starts
        for k in 0 to n - 1 loop -- Generate 1 bit at a time for G, f and Y
```

```
1
2           if (g = 0) then -- NOT USING GALOIS
3               fWire(128 + k) := fWire(k) xor fWire(k + 7) xor fWire(k + 38) xor fWire(70 + k) xor fWire(81 + k)
4   xor fWire(96 + k);
5
6               gWire(128 + k) := gWire(k) xor fWire(k) xor gWire(k + 26) xor gWire(k + 56) xor gWire(k + 91) xor
7   gWire(k + 96) xor
8                   (gWire(k + 3) and gWire(k + 67)) xor (gWire(k + 11) and gWire(k + 13)) xor (gWire(k + 17) and
9   gWire(k + 18)) xor
10                  (gWire(k + 27) and gWire(k + 59)) xor (gWire(k + 40) and gWire(k + 48)) xor (gWire(k + 61) and
11  gWire(k + 65)) xor
12                  (gWire(k + 68) and gWire(k + 84)) xor (gWire(k + 88) and gWire(k + 92) and gWire(k + 93) and
13  gWire(k + 95)) xor
14                  (gWire(k + 22) and gWire(k + 24) and gWire(k + 25)) xor (gWire(k + 70) and gWire(k + 78) and
15  gWire(k + 82));
16          else -- USING GALOIS
17              if (n = 16) then -- 16 PARA
18                  fWire(128 + k) := fWire(k) xor fWire(k + 7) xor fWire(k + 38) xor fWire(k + 70);--f127 = s0 ? s7 ?
19  s38 ? s70
20
21                  --fWire(k+112 + tconst3) xor
22                  fWire(128 + k + 16) := fWire(k + 112 + tconst3) xor fWire(k + 65 + tconst) xor fWire(k + 80 +
23  tconst);--f111 = s112 ? s65 ? s80
24                  gWire(128 + k)      := fWire(k) xor gWire(k) xor gWire(k + 56) xor (gWire(k + 3) and gWire(k +
25  67))
26                      xor (gWire(k + 11) and gWire(k + 13)) xor (gWire(k + 40) and gWire(k + 48)) xor
27                      (gWire(k + 22) and gWire(k + 24) and gWire(k + 25)) xor (gWire(k + 70) and gWire(k + 78) and
28  gWire(k + 82));
29
30                  gWire(128 + k + 16) := gWire(k + 112 + tconst3) xor gWire(k + 10 + tconst) xor gWire(k + 75 +
31  tconst)
32                      xor gWire(k + 80 + tconst) xor (gWire(k + 1 + tconst) and gWire(k + 2 + tconst))
33                      xor (gWire(k + 11 + tconst) and gWire(k + 43 + tconst)) xor (gWire(k + 45 + tconst)
34                      and gWire(k + 49 + tconst)) xor (gWire(k + 72 + tconst) and gWire(k + 76 + tconst)
35                      and gWire(k + 77 + tconst) and gWire(k + 79 + tconst)) xor (gWire(k + 68 + tconst) and gWire(k
36  + 52 + tconst));
37
38              elsif (n = 8) then
39                  fWire(128 + k)      := fWire(k + 0) xor fWire(k + 7) xor fWire(k + 38);
40                  fWire(128 + k + 8)  := fWire(k + 120) xor fWire(k + 62);
41                  fWire(128 + k + 16) := fWire(k + 112) xor fWire(k + 65);
42                  fWire(128 + k + 24) := fWire(k + 104) xor fWire(k + 72);
43
44                  gWire(128 + k)      := fWire(k + 0) xor gWire(k + 0) xor (gWire(k + 3) and gWire(k + 67)) xor
45  (gWire(k + 88) and gWire(k + 92) and gWire(k + 93) and gWire(k + 95));
46                  gWire(128 + k + 8)  := gWire(k + 120) xor (gWire(k + 9) and gWire(k + 10)) xor (gWire(k + 3)
47  and gWire(k + 5)) xor (gWire(k + 32) and gWire(k + 40)) xor (gWire(k + 60) and gWire(k + 76));
48                  gWire(128 + k + 16) := gWire(k + 112) xor gWire(k + 10) xor gWire(k + 40) xor (gWire(k + 11)
49  and gWire(k + 43)) xor gWire(k + 75) xor (gWire(k + 6) and gWire(k + 8) and gWire(k + 9));
50                  gWire(128 + k + 24) := gWire(k + 104) xor gWire(k + 72) xor (gWire(k + 37) and gWire(k + 41))
51  xor (gWire(k + 46) and gWire(k + 54) and gWire(k + 58));
52
53              elsif (n <= 4) then
54                  fWire(128 + k)      := fWire(k + 0) xor fWire(k + 7);
55                  fWire(128 + k + 4)  := fWire(k + 124) xor fWire(k + 34);
56                  fWire(128 + k + 8)  := fWire(k + 120) xor fWire(k + 62);
```

```
                fWire(128 + k + 12) := fWire(k + 116) xor fWire(k + 69);
                fWire(128 + k + 16) := fWire(k + 112) xor fWire(k + 80);

                if (n = 4) then
                   gWire(128 + k)      := fWire(k + 0) xor gWire(k + 0) xor (gWire(k + 3) and gWire(k + 67));
                   gWire(128 + k + 4)  := gWire(k + 124) xor gWire(k + 22) xor gWire(k + 52) xor (gWire(k + 23)
and gWire(k + 55));
                   gWire(128 + k + 8)  := gWire(k + 120) xor (gWire(k + 9) and gWire(k + 10)) xor (gWire(k + 3)
and gWire(k + 5));
                   gWire(128 + k + 12) := gWire(k + 116) xor (gWire(k + 70) and gWire(k + 66) and gWire(k +
58));
                   gWire(128 + k + 16) := gWire(k + 112) xor (gWire(k + 6) and gWire(k + 8) and gWire(k + 9));
                   gWire(128 + k + 20) := gWire(k + 108) xor (gWire(k + 68) and gWire(k + 72) and gWire(k +
73) and gWire(k + 75));
                   gWire(128 + k + 24) := gWire(k + 104) xor gWire(k + 72) xor (gWire(k + 37) and gWire(k +
41));
                   gWire(128 + k + 28) := gWire(k + 100) xor (gWire(k + 40) and gWire(k + 56)) xor gWire(k +
63) xor (gWire(k + 12) and gWire(k + 20));
                elsif (n = 2) then

                   gWire(128 + k)     := gWire(k + 0) xor fWire(k + 0);
                   gWire(128 + k + 2) := gWire(k + 126) xor (gWire(k + 1) and gWire(k + 65));
                   gWire(128 + k + 4) := gWire(k + 124) xor (gWire(k + 57) and gWire(k + 61));
                   gWire(128 + k + 6) := gWire(k + 122) xor (gWire(k + 5) and gWire(k + 7));
                   gWire(128 + k + 8) := gWire(k + 120) xor (gWire(k + 9) and gWire(k + 10));
                   -- g119 to g115, extra space here.
                   gWire(128 + k + 12) := gWire(k + 116) xor (gWire(k + 15) and gWire(k + 47));
                   gWire(128 + k + 14) := gWire(k + 114) xor gWire(k + 12);
                   gWire(128 + k + 16) := gWire(k + 112) xor (gWire(k + 6) and gWire(k + 8) and gWire(k + 9));
                   gWire(128 + k + 18) := gWire(k + 110) xor gWire(k + 73);
                   gWire(128 + k + 20) := gWire(k + 108) xor (gWire(k + 62) and gWire(k + 58) and gWire(k +
50));
                   gWire(128 + k + 22) := gWire(k + 106) xor (gWire(k + 18) and gWire(k + 26));
                   gWire(128 + k + 24) := gWire(k + 104) xor gWire(k + 72);
                   gWire(128 + k + 26) := gWire(k + 102) xor gWire(k + 30);
                   gWire(128 + k + 28) := gWire(k + 100) xor (gWire(k + 40) and gWire(k + 56));
                   gWire(128 + k + 30) := gWire(k + 98) xor (gWire(k + 58) and gWire(k + 62) and gWire(k + 63)
and gWire(k + 65));
                end if;

             end if;

          end if; -- Done generating f and g for alla para versions
          -- Generating Y for all para versions + when and when not using y para.

          if (yt = 0) then -- NOT USING Y transform

             yWire(k) := (gWire(k + 12) and fWire(k + 8)) xor (fWire(k + 13) and fWire(k + 20)) xor
             (gWire(k + 95) and fWire(k + 42)) xor (fWire(k + 60) and fWire(k + 79)) xor
             (gWire(k + 12) and gWire(95 + k) and fWire(94 + k)) xor -- This is the h function (to the left and up)
             fWire(k + 93) xor gWire(k + 2) xor gWire(k + 15) xor gWire(k + 36) xor gWire(k + 45) xor
             gWire(k + 64) xor gWire(k + 73) xor gWire(k + 89);

          else -- USING y Transform.

             if (n <= 16) then
```

```
                yWire(k) := (gWire(k + 12) and fWire(k + 8)) xor (fWire(k + 13) and fWire(k + 20)) xor gWire(k +
15) xor
                   (gWire(k + 12) and gWire(k + 95) and fWire(k + 94)) xor gWire(k + 2) xor gWire(k + 64);
                yWire2(k) := (gWire(k + 79) and fWire(k + 26)) xor fWire(k + 77) xor gWire(k + 73) xor gWire(k
+ 57) xor
                   (fWire(k + 44) and fWire(k + 63)) xor gWire(k + 20) xor gWire(k + 29);
                   --(fWire(k + 44 + tconst2) and fWire(k + 63 + tconst2)) xor gWire(k + 20 + tconst2) xor gWire(k
+ 29 + tconst2);

                PipelineNext(k) <= yWire2(k);

                if (iGrainYEnable = '0') then -- Pipeline
                   yWire(k) := yWire(k) xor PipelineReg(k);
                end if;

                gWire(128 + k + 16) := gWire(128 + k + 16) xor (yWire2(k) and enableReg);--iGrainYEnable);
                fWire(128 + k + 16) := fWire(128 + k + 16) xor (yWire2(k) and enableReg);
             end if;

          end if;

          gWire(128 + k) := gWire(128 + k) xor (yWire(k) and iGrainYEnable);
          fWire(128 + k) := fWire(128 + k) xor (iGrainKey(k) and iGrainFP1) xor (yWire(k) and iGrainYEnable);
-- iGrainFP1 and iGrainYEnable will never be one at the same time.

          if (iGrainStart = '1') then -- Loading which value is shifted in
             grainLfsrNext(128 - n + k) <= fWire(128 + k); --xor (iGrainIv(k) AND (NOT iGrainStart));
             grainNfsrNext(128 - n + k) <= gWire(128 + k); -- xor (iGrainKey(k) AND (NOT iGrainStart))

             if (g = 0) then -- Not doing galois

             else
                if (n = 16) then -- 112 96
                   grainLfsrNext(128 - n + k - 16) <= fWire(128 + k + 16); -- First feedback function goes here.
                   grainNfsrNext(128 - n + k - 16) <= gWire(128 + k + 16);
                elsif (n = 8) then
                   grainLfsrNext(128 - n + k - 8)  <= fWire(128 + k + 8);
                   grainLfsrNext(128 - n + k - 16) <= fWire(128 + k + 16);
                   grainLfsrNext(128 - n + k - 24) <= fWire(128 + k + 24);
                   grainNfsrNext(128 - n + k - 8)  <= gWire(128 + k + 8);
                   grainNfsrNext(128 - n + k - 16) <= gWire(128 + k + 16);
                   grainNfsrNext(128 - n + k - 24) <= gWire(128 + k + 24);
                elsif (n <= 4) then
                   grainLfsrNext(128 - n + k - 4)  <= fWire(128 + k + 4);
                   grainLfsrNext(128 - n + k - 8)  <= fWire(128 + k + 8);
                   grainLfsrNext(128 - n + k - 12) <= fWire(128 + k + 12);
                   grainLfsrNext(128 - n + k - 16) <= fWire(128 + k + 16);

                   if (n = 4) then
                      grainNfsrNext(128 - n + k - 4)  <= gWire(128 + k + 4);
                      grainNfsrNext(128 - n + k - 8)  <= gWire(128 + k + 8);
                      grainNfsrNext(128 - n + k - 12) <= gWire(128 + k + 12);
                      grainNfsrNext(128 - n + k - 16) <= gWire(128 + k + 16);
                      grainNfsrNext(128 - n + k - 20) <= gWire(128 + k + 20);
                      grainNfsrNext(128 - n + k - 24) <= gWire(128 + k + 24);
                      grainNfsrNext(128 - n + k - 28) <= gWire(128 + k + 28);
```

```
                elsif (n = 2) then
                    grainNfsrNext(128 - n + k - 2)  <= gWire(128 + k + 2);
                    grainNfsrNext(128 - n + k - 4)  <= gWire(128 + k + 4);
                    grainNfsrNext(128 - n + k - 6)  <= gWire(128 + k + 6);
                    grainNfsrNext(128 - n + k - 8)  <= gWire(128 + k + 8);
                    grainNfsrNext(128 - n + k - 12) <= gWire(128 + k + 12);
                    grainNfsrNext(128 - n + k - 14) <= gWire(128 + k + 14);
                    grainNfsrNext(128 - n + k - 16) <= gWire(128 + k + 16);
                    grainNfsrNext(128 - n + k - 18) <= gWire(128 + k + 18);
                    grainNfsrNext(128 - n + k - 20) <= gWire(128 + k + 20);
                    grainNfsrNext(128 - n + k - 22) <= gWire(128 + k + 22);
                    grainNfsrNext(128 - n + k - 24) <= gWire(128 + k + 24);
                    grainNfsrNext(128 - n + k - 26) <= gWire(128 + k + 26);
                    grainNfsrNext(128 - n + k - 28) <= gWire(128 + k + 28);
                    grainNfsrNext(128 - n + k - 30) <= gWire(128 + k + 30);
                end if;

            end if;

        end if;
    end if;
  end loop;

  oGrainY <= yWire(0 to n-1);

end process;
-------------------------------------------------------------------------------

end behavioral;
```

Ακολουθεί το περιεχόμενο του αρχείου accumulator.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity accumulator is
    generic (
        n1 : integer; -- Parallellization/unrolling for accumulator. (half of the grain)
        n2 : integer -- Parallellization/unrolling for grain. n1*2
    );
    port (
        iAccumStart      : in STD_LOGIC;
        iAccumY          : in STD_LOGIC_VECTOR(0 to n2 - 1); -- Send n2 bits. The last half only used for
loading.
        oAccumRegFeedback : out STD_LOGIC_vector(63 downto 0); -- Basically tag.

        iAccumMsg        : in STD_LOGIC_VECTOR(0 to n1 - 1);
        iAccumClk        : in STD_LOGIC;
        iAccumReset      : in STD_LOGIC
    );
end accumulator;
```

```
1
2   architecture Behavioral of accumulator is
3      ---------------------------------------------------------------------------
4
5      -- Define the two registers: LFSR & NFSR (You can't use the same signal for input end Behavioral;
6      signal AccumReg      : STD_LOGIC_vector(63 downto 0); -- Current value
7      signal AccumNext     : STD_LOGIC_vector(63 downto 0); -- Next value
8      signal AccumShiftReg  : STD_LOGIC_vector(63 downto 0); -- Current value
9      signal AccumShiftNext : STD_LOGIC_vector(63 downto 0); -- Next value
10
11     -- Since the tag is defined as ti = a^i_L+1, 0 <= i <= 31.
12     -- It's better if the shift and accum are encoded that way. Which is why downto is used instead of to.
13     --constant n1 : integer := 32; -- Accum parallellization. Supports up to 64
14     --constant n2 : integer := n1*2; -- real parallelization.
15
16     ---------------------------------------------------------------------------
17  begin
18     ---------------------------------------------------------------------------
19     seq : process (iAccumClk, iAccumReset)
20     begin
21        if (iAccumReset = '1') then
22           AccumReg      <= (others => '0');
23           AccumShiftReg <= (others => '0');
24        elsif (rising_edge(iAccumClk)) then
25           AccumReg      <= AccumNext;
26           AccumShiftReg <= AccumShiftNext;
27        end if;
28     end process;
29
30     ---------------------------------------------------------------------------
31
32     comb : process (iAccumStart, iAccumY,
33        iAccumReset, AccumReg, AccumShiftReg,
34        iAccumMsg)
35        variable AccumRegNewval    : STD_LOGIC_VECTOR(63 downto 0); -- New generated values for
36  accum.
37        variable AccumShiftRegWire : STD_LOGIC_VECTOR(63 + (n1 - 1) downto 0); -- Wire for ShiftReg |
38  n1-1 YAccum bits, that are required in accumlogic.
39     begin
40        AccumShiftRegWire(63 + (n1 - 1) downto n1 - 1) := AccumShiftReg; -- Rest n1-2.
41        if (n1 > 1) then -- Needs to used the YAccum value in updating.
42
43           AccumShiftRegWire := AccumShiftReg & iAccumY(0 to n1 - 2);
44           -- AccumShiftRegWire(n1-2 downto 0) := iAccumY(n1-1 downto 1); -- Only n1-1 values are of the
45  yAccum is required for the wire
46
47        else
48           AccumShiftRegWire := AccumShiftReg; --
49        end if;
50
51        for k in 0 to 63 loop
52           AccumRegNewval(k) := AccumReg(k);
53           for a in 0 to n1 - 1 loop -- Accum logic for the parallellization.
54              AccumRegNewval(k) := AccumRegNewval(k) xor (iAccumMsg(n1 - 1 - a) and
55  AccumShiftRegWire(a + k)); --
56           end loop;
```

```
        end loop;
        --iAccumMsg

        -- It's mentioned in the paper that ti = a^i_l+1, 0 <= i <= 31. Which means we need to reverse the bits for
the output.
        AccumNext        <= AccumRegNewval;
        oAccumRegFeedback <= AccumReg;
        if (iAccumStart = '1') then -- Normal mode n/2 shift.
            AccumShiftNext <= AccumShiftReg(63 - n1 downto 0) & iAccumY(0 to (n1 - 1)); -- Does
accumulation with n2/2 bits

        else -- Loading. n shift
            if (n2 = 64) then
                AccumShiftNext <= iAccumY; -- Loads all values directly into the reg.

            elsif (n2 = 1) then
                AccumShiftNext <= AccumShiftReg; -- paused.

            else
                AccumShiftNext <= AccumShiftReg(63 - n2 downto 0) & iAccumY(0 to (n2 - 1)); -- Loads all the Y
values sent in. n2 bits.

            end if;
        end if;

    end process;
    -------------------------------------------------------------------------------
end behavioral;
```

Στη συνέχεια το περιεχόμενο του αρχείου AuthPipeline.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity authPipeline is
    generic (
        n     : integer;
        ndiv2 : integer;
        ia    : integer -- Isolation or not
    );
    port (
        iAuthPipelineMsgIn        : in std_logic_vector (0 to ndiv2 - 1);
        oAuthPipelineMsgReg       : out std_logic_vector (0 to ndiv2 - 1);
        iAuthPipelineYAccum       : in std_logic_vector (0 to n - 1);
        oAuthPipelineYAccumReg    : out std_logic_vector (0 to n - 1);
        iAuthPipelineAccumStart   : in std_logic;
        oAuthPipelineAccumStartReg : out std_logic;
        iAuthPipelineClk          : in std_logic;
        iAuthPipelineReset        : in std_logic
    );
end authPipeline;
```

```
1
2   architecture Behavioral of authPipeline is
3
4      signal MsgReg, MsgNext, MsgReg2, MsgNext2        : std_logic_vector(0 to ndiv2 - 1); -- Current value
5      signal YAccumReg, YAccumNext                     : std_logic_vector(0 to n - 1); -- Current value
6      signal startReg, startNext, startReg2, startNext2 : std_logic;
7      ----------------------------------------------------------------------------
8   begin
9      ----------------------------------------------------------------------------
10     seq : process (iAuthPipelineClk, iAuthPipelineReset)
11     begin
12        if (iAuthPipelineReset = '1') then
13           MsgReg    <= (others => '0');
14           YAccumReg <= (others => '0');
15           startReg  <= '0';
16           MsgReg2   <= (others => '0');
17           startReg2 <= '0';
18        elsif (rising_edge(iAuthPipelineClk)) then
19           YAccumReg <= YAccumNext;
20           MsgReg    <= MsgNext;
21           startReg  <= startNext;
22           MsgReg2   <= MsgNext2;
23           startReg2 <= startNext2;
24        end if;
25     end process;
26
27     ----------------------------------------------------------------------------
28     comb : process (iAuthPipelineMsgIn, iAuthPipelineYAccum, MsgReg, MsgReg2, YAccumReg, startReg,
29  startReg2, iAuthPipelineAccumStart)
30     begin
31        YAccumNext <= iAuthPipelineYAccum;
32        MsgNext    <= iAuthPipelineMsgIn;
33        startNext  <= iAuthPipelineAccumStart;
34        MsgNext2   <= MsgReg;
35        startNext2 <= startReg;
36
37        if (ia = 1) then
38           oAuthPipelineMsgReg       <= MsgReg;
39           oAuthPipelineYAccumReg    <= YAccumReg;
40           oAuthPipelineAccumStartReg <= startReg;
41
42        else -- No isolation.
43           oAuthPipelineMsgReg       <= iAuthPipelineMsgIn;
44           oAuthPipelineYAccumReg    <= iAuthPipelineYAccum;
45           oAuthPipelineAccumStartReg <= iAuthPipelineAccumStart;
46        end if;
47
48     end process;
49     ----------------------------------------------------------------------------
50  end Behavioral;
51
```

52

53 Παρομοίως το περιεχόμενο του αρχείου ClockDiv.vhd

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.all;
4   use IEEE.NUMERIC_STD.all;
5   use IEEE.math_real.all;
6
7   entity clock_div is
8      generic (
9         ctype     : integer;
10        clkdivnum : integer -- How much clock division. Put to 1 to disable.
11     );
12     port (
13        clk    : in STD_LOGIC;
14        reset  : in STD_LOGIC;
15        clkdiv : out STD_LOGIC
16     );
17  end clock_div;
18
19  architecture Behavioral of clock_div is
20
21     constant CounterBits         : integer := integer(ceil(log2(real(clkdivnum)))); -- Get the highest counter
22  needed.
23     signal shiftReg, shiftRegNext : unsigned(0 to CounterBits - 1); -- 31 STD_LOGIC_VECTOR. For counting
24  the clock cycles in each stage.
25
26  begin
27     seq : process (clk, reset)
28     begin
29        if (ctype = 1) then
30           if (reset = '1') then
31              shiftReg    <= (others => '0');
32              shiftReg(0) <= '1'; -- When using original for new one.
33           elsif (rising_edge(clk)) then
34              shiftReg <= shiftRegNext;
35           end if;
36        end if;
37     end process;
38     comb : process (shiftReg, clk)
39     begin
40        if (ctype = 1) then
41           shiftRegNext <= shiftReg + 1;
42           clkdiv       <= shiftReg(0);
43        else
44           clkdiv <= clk; -- Just output normal clock otherwise.
45        end if;
46
47     end process;
48
49  end Behavioral;
50
```

51

52 ακολουθεί το περιεχόμενο του αρχείου Controller.vhd

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.NUMERIC_STD.all;
4
5   use IEEE.math_real.all;
6
7   ---------------------------------------------------------------------------
8   entity controller is
9     generic (
10      n        : integer;--;
11      ndiv2    : integer;
12      ctype    : integer;
13      clkdivnum : integer -- How much clock division.
14
15
16      --b1 : INTEGER -- How many bits you wanna load.
17    );
18    port (
19      iControllerEnable     : in std_logic; -- Enable signal from top. Paused if zero.
20
21      oControllerYFlag      : out std_logic; -- If keystream is available just goes to output
22      oControllerYEnable    : out std_logic; -- Input to F and G, is in init stage if 1. coming from contr
23      oControllerMsgNewval  : out std_logic_vector(0 to ndiv2 - 1); -- Input to accumlogic from controlle
24      iControllerMsgIn      : in std_logic_vector(0 to ndiv2 - 1); -- Input to controller from Top.
25
26      oControllerAccumStart  : out std_logic; -- (Decides whether or not accum should be used or not. Will be
27      oControllerGrainStart  : out std_logic; -- Controls if the grain runs. Is automatically stopped during
28      oControllerFP1        : out std_logic; -- Sending out if we should do FP1
29      iControllerYFeedback   : in std_logic_vector(0 to n - 1); -- Get the Y values for outputting from the
30      oControllerYAccum     : out std_logic_vector(0 to n - 1); -- Output Y that goes to accum. When it's
31      oControllerYOut       : out std_logic_vector(0 to ndiv2 - 1);
32
33      iControllerCryptMode   : std_logic; -- 1 = decryption. 0 = Encryption
34      iControllerCryptEnable : std_logic; -- Enable/disable encryption
35      iControllerReset      : in std_logic;
36      iControllerClk        : in std_logic; -- For the counter register so it shifts 1 every 32 clock cycles.
37      iControllerClk2       : in std_logic -- Used in 1 para modified/new controller.
38    );
39
40  end controller;
41
42  ---------------------------------------------------------------------------
43
44  architecture Behavioral of controller is
45    type stateType is (RESET, LOAD, INIT, ACCUMLOAD, NORMAL);
46    signal state, nextState : stateType;
47
48    -- Key/iv must be n or else FP1 doesn't work.
49    constant ees                          : integer := 1;
50    constant LoadClock                    : integer := (128/n); -- Clock cycles for accumulator loading and
51  also grain loading.
52    constant InitClock                    : integer := (256/n); -- For initilization.
53    constant TotalClock                   : integer := LoadClock * 2 + InitClock; -- Total amount of clock
54  cycles.
55    constant CounterBits                  : integer := integer(ceil(log2(real(InitClock)))); -- Get the highest
56  counter needed.
```

```
1
2       constant Index1                         : integer := (LoadClock/(clkdivnum)) - 1; -- Register that is 1 when
3   init (3
4       constant Index2                         : integer := ((LoadClock + InitClock)/clkdivnum) - 1; -- Register that
5   is 1 when accumload(11. 2
6       constant Index3                         : integer := ((2 * LoadClock + InitClock)/clkdivnum) - 1; -- When all
7   finished. (15
8
9       signal controllerCounterReg, controllerCounterNext   : unsigned(0 to Index3); -- Different counter size
10  depending on the controller
11      signal controllerCounterReg2, controllerCounterNext2 : unsigned(0 to (CounterBits - 1)); -- Different counter
12  size depending on the controller
13
14      signal grainStartReg, grainStartNext            : std_logic;
15      signal FP1Reg, FP1Next                          : std_logic;
16      signal YEnableReg, YEnableNext                  : std_logic;
17
18      -- For 1 para only
19      signal modeReg, modeNext : std_logic; -- 1 = doing accum, 0 = keystream for the 1 para version which cant
20  do both at the same time.
21      signal prevReg, prevNext : std_logic; -- Used to store the keystream xor ciphertext (ciphertext will be
22  message then), which will be used during decrypt.
23
24      ---------------------------------------------------------------------------
25  begin
26      ---------------------------------------------------------------------------
27
28      seq : process (iControllerClk, iControllerReset)
29      begin
30          if (iControllerReset = '1') then
31              state               <= RESET;
32              controllerCounterReg2 <= (others => '0');
33              controllerCounterReg  <= (others => '0');
34              grainStartReg       <= '0';
35              FP1Reg              <= '0';
36              YEnableReg          <= '0';
37          elsif (rising_edge(iControllerClk)) then
38              state               <= nextState;
39              controllerCounterReg  <= controllerCounterNext;
40              controllerCounterReg2 <= controllerCounterNext2;
41              grainStartReg       <= grainStartNext;
42              FP1Reg              <= FP1Next;
43              YEnableReg          <= YEnableNext;
44
45          end if;
46      end process;
47
48      seq2 : process (iControllerClk2, iControllerReset) -- 1 para with modified controller requires the normal
49  clock to run the other two registers.
50      begin
51          if (n = 1) then
52              if (iControllerReset = '1') then
53                  modeReg <= '0';
54                  prevReg <= '0';
55              elsif (rising_edge(iControllerClk2)) then
56                  modeReg <= modeNext;
```

```
1               prevReg <= prevNext;
2             end if;
3           end if;
4       end process;
5       ---------------------------------------------------------------------------
6       comb : process (state, iControllerMsgIn, FP1Reg, grainStartReg, YEnableReg, controllerCounterReg,
7   iControllerCryptMode,
8             iControllerCryptEnable, controllerCounterReg2, iControllerEnable, iControllerYFeedback, modeReg,
9   prevReg) -- iControllerMsgFlag
10          variable yAccumWire : std_logic_vector(0 to n - 1); -- Wire for
11          variable yOutWire   : std_logic_vector(0 to ndiv2 - 1);
12      begin
13          ----------------------------------
14
15          -- Code used in both controllers.
16          yAccumWire := (others => '0');
17          yOutWire   := (others => '0');
18          if (n > 1) then
19              for k in 0 to ndiv2 - 1 loop
20                  yAccumWire(k) := iControllerYFeedback(k * 2 + 1);
21                  yOutWire(k)   := iControllerYFeedback(k * 2);
22              end loop;
23          end if;
24
25          if (ctype = 0) then
26              grainStartNext        <= grainStartReg;
27              FP1Next               <= FP1Reg;
28              YEnableNext           <= YEnableReg;
29              oControllerGrainStart  <= grainStartReg;
30              oControllerFP1        <= FP1Reg;
31              oControllerYEnable     <= YEnableReg;
32
33              oControllerYFlag      <= '0'; -- If keystream is available just goes to output
34
35              nextState             <= state;
36              controllerCounterNext2 <= controllerCounterReg2;
37              oControllerMsgNewval   <= (others => '0');
38              oControllerAccumStart  <= '0'; -- Loading.
39
40              oControllerYAccum      <= (others => '0');
41
42              oControllerYOut       <= yOutWire;
43
44              modeNext              <= modeReg;
45              prevNext              <= prevReg;
46
47              ----------------------------------
48              case (state) is
49
50                  when RESET =>
51                      controllerCounterNext2 <= (others => '0');
52
53                      grainStartNext        <= '0';
54                      FP1Next               <= '0';
55                      YEnableNext           <= '0';
56
```

```
1                   if (iControllerEnable = '1') then
2                      nextState <= LOAD;
3
4                   end if;
5
6                   ----------------------------------
7                when LOAD =>
8
9                   controllerCounterNext2 <= controllerCounterReg2 + 1;
10                  if (controllerCounterReg2 = LoadClock - 2) then -- All loading clock cycles.
11                     nextState            <= INIT;
12                     grainStartNext       <= '1';
13                     YEnableNext          <= '1';
14                     controllerCounterNext2 <= (others => '0');
15                  end if;
16
17                  ----------------------------------
18               when INIT =>
19
20                  controllerCounterNext2 <= controllerCounterReg2 + 1;
21                  if (controllerCounterReg2 = InitClock - 1) then
22                     nextState            <= ACCUMLOAD;
23                     FP1Next              <= '1';
24                     YEnableNext          <= '0';
25                     controllerCounterNext2 <= (others => '0');
26                  end if;
27
28                  ----------------------------------
29               when ACCUMLOAD =>
30                  -- Run for 32 clock with message sent into logic = 0.
31                  -- Then we set message to one to put into accum
32                  -- Then message zero again for 32 cloc => loading of accum & shift reg.
33
34                  oControllerYFlag      <= '1';
35                  oControllerMsgNewval  <= iControllerMsgIn;
36
37                  controllerCounterNext2 <= controllerCounterReg2 + 1;
38
39                  if (n = 1) then
40                     oControllerAccumStart <= '1'; -- 1 para has paused or no pause. 1 means we shift in 1 value.
41                  else
42                     oControllerAccumStart <= '0'; -- n values shifted in during loading when accum 0.
43                  end if;
44                  oControllerYAccum <= iControllerYFeedback; -- Loads all 32 values.
45                  if (controllerCounterReg2 = LoadClock/2) then -- After half we put message to 1 to just load the
46    shift register values into the accumulator.
47
48                  end if;
49
50                  if (controllerCounterReg2 = LoadClock - 1) then -- All bits are shifted into the shift register.
51                     nextState            <= NORMAL;
52                     FP1Next              <= '0';
53                     controllerCounterNext2 <= (others => '0');
54                  end if;
55
56                  ----------------------------------
```

```
1               when NORMAL =>
2
3                   oControllerYFlag <= '1'; -- If keystream is available just goes to output
4
5
6                   if (n = 1) then
7                       modeNext            <= not modeReg; -- Every second clock cycle this occurs.
8                       oControllerAccumStart <= modeReg; -- Only active every second clock cycle.
9                       if (modeReg = '0') then -- Always send zero when everything is paused.
10                          yOutWire(0) := iControllerYFeedback(0); -- Keystream bit.
11
12                      else
13                          yAccumWire(0) := iControllerYFeedback(0); -- Mac bit.
14                      end if;
15                  else
16
17                      oControllerAccumStart <= '1'; --
18                  end if;
19
20                  if (n = 1) then
21                      if (iControllerCryptEnable = '0') then
22
23                          prevNext <= iControllerMsgIn(0); -- Needs to be the message when cryptenable is off.
24                          yOutWire := (others => '0'); -- No encryption/decryption in the current clock. YOut will be
25  just 0 xor Message = message.
26                      else
27                          prevNext <= yOutWire(0) xor iControllerMsgIn(0); -- Will only be used when 1 para.
28                      end if;
29
30                  else
31                      if (iControllerCryptEnable = '0') then
32
33
34                          yOutWire := (others => '0');
35                      end if;
36
37                  end if;
38
39                  -- Encrypt and decrypt.
40                  if (iControllerCryptMode = '1') then -- Decryption.
41                      if (n = 1) then
42                          if (modeReg = '0') then
43                              oControllerMsgNewval(0) <= '0'; -- Message going to accum is 0 zero when accum is
44  paused.
45                          else
46                              oControllerMsgNewval(0) <= prevReg;
47                          end if;
48                      else
49                          oControllerMsgNewval <= yOutWire xor iControllerMsgIn; -- mi = KeyStream xor ci
50  (ciphertext send in)
51                      end if;
52                  else
53                      oControllerMsgNewval <= iControllerMsgIn;
54
55                  end if;
56
```

```
1                   -- Makes wires for the signals so that everything can be grouped together.
2                   oControllerYAccum <= yAccumWire; -- Ports the wires out of the design.
3                   oControllerYOut   <= yOutWire xor iControllerMsgIn;
4                   ------------------------------------
5
6                when others =>
7
8             end case; -- Ends all the states.
9          else -- Alternative controller.
10
11            -- 1 para: Register for saving message. modereg as accumstart? Then YOut should zero when modereg
12    is zero.
13            controllerCounterNext(0 to Index3) <= iControllerEnable & controllerCounterReg(0 to Index3 - 1);
14
15            oControllerYFlag                <= controllerCounterReg(Index2); --AND (NOT
16    controllerCounterReg2(511)); -- When 1 we are doing keystream generation otherwise doing accum. Always 1
17    during accumload. Note that in the other code we used a register value that was 0 when we did keystream.
18            oControllerYEnable              <= controllerCounterReg(Index1) and (not
19    controllerCounterReg(Index2)); -- When loading is done and when init is not finished, we YEnable.
20            oControllerGrainStart           <= controllerCounterReg(Index1);
21            oControllerFP1                  <= controllerCounterReg(Index2) and (not controllerCounterReg(Index3));
22
23            oControllerYOut                 <= (others => '0');
24            oControllerYAccum               <= (others => '0');
25
26            modeNext                        <= modeReg;
27            prevNext                        <= prevReg;
28
29            oControllerAccumStart           <= '0'; -- Zero by default. 1 in normal state for all paras. 1 in accumload
30    and then when accumulator is active in the normal state, for 1 para.
31
32
33            -- Muxes for the output signal as well as the yaccum signal going to the accumulator.
34            if (controllerCounterReg(Index3) = '1') then -- Normal state.
35               if (n = 1) then -- If 1 para
36                  modeNext          <= ((not modeReg) and controllerCounterReg(Index3)); -- Changes value
37    between 1 and 0 after we reach the normal state.
38                  oControllerAccumStart <= modeReg and controllerCounterReg(Index3);
39                  if (modeReg = '0') then -- Always send zero when everything is paused.
40                     yOutWire(0) := iControllerYFeedback(0); -- Keystream bit.
41
42                  else
43                     yAccumWire(0) := iControllerYFeedback(0); -- Mac bit.
44                  end if;
45               else
46                  oControllerAccumStart <= controllerCounterReg(Index3); -- 1 when in normal.
47               end if;
48
49               if (iControllerCryptEnable = '0') then
50
51                  yOutWire := (others => '0'); -- No encryption/decryption in the current clock.
52               else
53                  oControllerYOut <= yOutWire; -- XOR iControllerMsgIn; Every second bit.
54
55               end if;
56
```

```
            if (n = 1) then
                prevNext <= yOutWire(0) xor iControllerMsgIn(0); -- Used for decryption in 1 para to store the
plaintext (message) needed for next clock.
            end if;
            oControllerYAccum <= yAccumWire; -- Every other second bit.

        elsif (controllerCounterReg(Index2) = '1') then -- In accumload.
            oControllerYAccum <= iControllerYFeedback;
            if (n = 1) then
                oControllerAccumStart <= '1'; -- 1 para needs accumload active
            end if;
        end if;

        if (controllerCounterReg(Index3) = '1') then -- Adding encryption/decryption.
            if (iControllerCryptMode = '1') then -- Decryption.
                if (n = 1) then --
                    if (modeReg = '0') then
                        oControllerMsgNewval(0) <= '0'; -- Message going to accum is 0 zero when accum is paused.
                    else
                        oControllerMsgNewval(0) <= prevReg; -- The message is the plaintext generated last clock.
                    end if;
                else
                    oControllerMsgNewval <= yOutWire xor iControllerMsgIn; -- mi = KeyStream xor ci
(ciphertext send in)
                end if;
            else
                oControllerMsgNewval <= iControllerMsgIn;
            end if;

        else
            oControllerMsgNewVal <= iControllerMsgIn; -- Always just send in the message.
        end if;

    end if;

  end process;

end Behavioral;
```

Στη συνέχεια το περιεχόμενο του αρχείου GrainTop.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

----------------------------------------------------------------------------

entity grainTop is
  generic (
      n       : integer := 64; -- Parallelization level. When using the testbench ignore these values, These are to
be set when running synthesis without the testbench
```

```
        ndiv2    : integer := 32; -- For non-parallelized version, set this to one, otherwise n/2.
        g        : integer := 0; -- 1 if Galois transform, else 0. If using with testbench you only need to change those
values.
        yt       : integer := 0; -- 1 if using Y-transform, else 0
        ctype    : integer := 1; -- 1 for optimized controller, 0 for standard.
        clkdivnum : integer := 2; -- clkdivnum * n = 128 for best result
        ia       : integer := 1 -- 1 for authentication isolation, else 0.
    );
    port (
        oGraintopY          : out STD_LOGIC_VECTOR(0 to (ndiv2) - 1); -- Key stream out.
        iGraintopIv         : in STD_LOGIC_VECTOR(0 to n - 1); -- IV in.
        iGraintopKey        : in STD_LOGIC_vector(0 to n - 1); -- Key in
        iGraintopEnable     : in STD_LOGIC; -- Start
        iGraintopReset      : in STD_LOGIC;
        iGraintopClk        : in STD_LOGIC;
        oGraintopYFlag      : out STD_LOGIC; -- When keystream available.
        oGraintopTag        : out STD_LOGIC_VECTOR(63 downto 0); -- The tag if using authentication.
        iGraintopMsg        : in STD_LOGIC_VECTOR(0 to (ndiv2) - 1); -- Message
        iGraintopCryptMode  : STD_LOGIC; -- 1 = decryption. 0 = Encryption
    iGraintopCryptEnable : STD_LOGIC); -- Enable/disable encryption

end grainTop;

--------------------------------------------------------------------------------

architecture structural of grainTop is

    ----------------------------CREATE COMPONENTS-------------------------------------------------------
    component clock_div is
        generic (
            ctype    : integer;
            clkdivnum : integer
        );

        port (
            clk   : in STD_LOGIC;
            reset  : in STD_LOGIC;
            clkdiv : out STD_LOGIC
        );
    end component;

    component authPipeline is
        generic (
            n    : integer;
            ndiv2 : integer;
        ia   : integer); -- Isolation or not

        port (
            iAuthPipelineMsgIn        : in STD_LOGIC_VECTOR (0 to (ndiv2) - 1);
            oAuthPipelineMsgReg       : out STD_LOGIC_VECTOR (0 to (ndiv2) - 1);
            iAuthPipelineYAccum       : in STD_LOGIC_VECTOR (0 to n - 1);
            oAuthPipelineYAccumReg    : out STD_LOGIC_VECTOR (0 to n - 1);
            iAuthPipelineAccumStart   : in STD_LOGIC;
            oAuthPipelineAccumStartReg : out STD_LOGIC;
            iAuthPipelineClk          : in STD_LOGIC;
            iAuthPipelineReset        : in STD_LOGIC
```

```
1         );
2       end component;
3       component controller is
4         generic (
5           n        : integer;
6           ndiv2    : integer;
7           ctype    : integer;
8           clkdivnum : integer
9         );
10
11        port (
12          iControllerEnable     : in STD_LOGIC; -- Enable signal from top. Paused if zero.
13          oControllerYFlag      : out STD_LOGIC; -- Flag used for testbench to know when to send in messages
14    and get the output.
15          oControllerYEnable    : out STD_LOGIC; -- If y is feedback it's one.
16          oControllerMsgNewval  : out STD_LOGIC_VECTOR(0 to (ndiv2) - 1); -- Input to accumlogic from
17    controlle
18          iControllerMsgIn      : in STD_LOGIC_VECTOR(0 to (ndiv2) - 1); -- Input to controller from Top.
19          oControllerAccumStart : out STD_LOGIC; -- Decides whether or not accum should be used or not.
20          oControllerGrainStart : out STD_LOGIC; -- For controlling if the cipher is running.
21          oControllerFP1        : out STD_LOGIC; -- Sending out if we should do FP1
22          iControllerYFeedback  : in STD_LOGIC_VECTOR(0 to n - 1); -- Get the Y values for outputting.
23          oControllerYAccum     : out STD_LOGIC_VECTOR(0 to n - 1); -- Y output going to accum
24          oControllerYOut       : out STD_LOGIC_VECTOR(0 to (ndiv2) - 1); -- Y output going to out from the
25    design
26          iControllerCryptMode  : STD_LOGIC; -- 1 = decryption. 0 = Encryption
27          iControllerCryptEnable : STD_LOGIC; -- 1 = Enable, 0 = disable encryption in current clock
28          iControllerReset      : in STD_LOGIC;
29          iControllerClk        : in STD_LOGIC; -- For the counter register so it shifts 1 every 32 clock cycles.
30        iControllerClk2       : in STD_LOGIC); -- For the counter register so it shifts 1 every 32 clock cycles.
31      end component;
32
33
34
35
36      component grain is
37        generic (
38          n  : integer;
39          g  : integer; -- Galois if 1.
40        yt : integer); -- Y transform / pipeline if 1
41
42        port (
43          iGrainStart   : in STD_LOGIC; -- When 1 the cipher runs.
44          iGrainFP1     : in STD_LOGIC; -- Active when doing FP1
45          iGrainYEnable : in STD_LOGIC;
46          iGrainIv      : in STD_LOGIC_VECTOR(0 to n - 1);
47          iGrainKey     : in STD_LOGIC_VECTOR(0 to n - 1);
48          oGrainY       : out STD_LOGIC_VECTOR(0 to n - 1);
49          iGrainReset   : in STD_LOGIC;
50          iGrainClk     : in STD_LOGIC
51        );
52      end component;
53
54
55      component accumulator is
56        generic (
```

```
1             n1 : integer; -- Parallellization/unrolling for accumulator. (half of the grain)
2         n2 : integer); -- Parallellization/unrolling for grain. n1*2
3
4         port (
5             iAccumStart      : in STD_LOGIC;
6             iAccumY          : in STD_LOGIC_VECTOR(0 to n2 - 1); -- Send n2 bits. The last half only used for
7     loading.
8             oAccumRegFeedback : out STD_LOGIC_vector(0 to 63); -- Basically tag.
9             iAccumMsg        : in STD_LOGIC_VECTOR(0 to n1 - 1);
10            iAccumClk        : in STD_LOGIC;
11            iAccumReset      : in STD_LOGIC
12        );
13    end component;
14    ---------------------------------------------------------------------------------
15
16    -- Define the signals
17
18    signal graintopLfsrFeedback         : STD_LOGIC_vector(0 to 127); -- Output from grain's LFSR going to F's
19    input
20    signal graintopLfsrIn               : STD_LOGIC_VECTOR(0 to n - 1); -- Input for grain's LFSR coming from
21    controller (either Iv or LfsrIn, depending on state)
22    signal graintopLfsrNewval           : STD_LOGIC_VECTOR(0 to n - 1); -- Output coming from F going to
23    controller input
24
25    signal graintopNfsrFeedback         : STD_LOGIC_vector(0 to 127); -- Output from NFSR going to G's input
26    signal graintopNfsrIn               : STD_LOGIC_VECTOR(0 to n - 1); -- Input for grain's NFSR from
27    controller. (either key or NfsrIn, depending on state)
28    signal graintopNfsrNewval           : STD_LOGIC_VECTOR(0 to n - 1); -- Output coming from G going to
29    controller input
30    signal graintopYFeedback            : STD_LOGIC_VECTOR(0 to n - 1); -- Input to F and G containing Y,
31    Used during INIT STAGE.
32    signal graintopYEnable              : STD_LOGIC; -- Input to F and G, is in init stage if 1 coming from
33    controller.
34    signal graintopYFlag                : STD_LOGIC; -- Output from controller to notify when keystream is ready.
35    signal graintopAccumlogicRegFeedback : STD_LOGIC_vector(63 downto 0); -- Output from accumulator
36    containing the accum register values
37
38    signal graintopMsgNewval            : STD_LOGIC_VECTOR(0 to (ndiv2) - 1); -- Input to accumlogic from
39    controller, (Will be MsgIn normally, but is modified during loading to save hardware)
40    signal graintopAccumStart           : STD_LOGIC; -- (Decides whether or not accum should be used or not.
41    Will be Enable or 0 when y bit is keystream)
42    signal grainTopGrainStart           : STD_LOGIC; -- Controls if the grain runs. Is automatically stopped
43    during accum
44    signal grainTopFP1                  : STD_LOGIC; -- Controls if the grain runs. Is automatically stopped during
45    accum
46
47    signal grainTopYAccum               : STD_LOGIC_VECTOR(0 to n - 1); -- Output Y that goes to accum.
48    When it's turned on.
49    signal grainTopYOut                 : STD_LOGIC_VECTOR(0 to (ndiv2) - 1); -- Output Y that goes to ut
50    every second bit
51
52    signal grainTopMsgReg               : STD_LOGIC_VECTOR(0 to (ndiv2) - 1); -- Pipeline auth stage to
53    sepearate them
54    signal grainTopYAccumReg            : STD_LOGIC_VECTOR(0 to n - 1);
55    signal grainTopAccumStartReg        : STD_LOGIC;
56
```

```
1       signal designClk                : STD_LOGIC;
2       ----------------------------------------------------------------------------
3
4   begin
5       clockdivInst : clock_div
6           generic map(
7               ctype     => ctype,
8               clkdivnum => clkdivnum -- How much clock division. Put to 1 to disable.
9           )
10
11          port map(
12              clk    => iGraintopClk,
13              reset  => iGraintopReset,
14              clkdiv => designClk
15          );
16
17      grainInst : grain
18          generic map(
19              n  => n,
20              g  => g,
21              yt => yt -- Y transform / pipeline if 1
22          )
23
24          port map(
25              iGrainStart   => grainTopGrainStart, -- When 1 the cipher runs.
26              iGrainFP1     => grainTopFP1, -- Active when doing FP1
27              iGrainYEnable => grainTopYEnable,
28              iGrainIv      => iGrainTopIv,
29              iGrainKey     => iGrainTopKey,
30              oGrainY       => grainTopYFeedback,
31              iGrainReset   => iGraintopReset,
32              iGrainClk     => iGraintopClk
33          );
34
35
36
37
38      contInst : controller
39          generic map(
40              n         => n, -- Parallellization/unrolling for accumulator. (half of the grain)
41              ndiv2     => ndiv2,
42              ctype     => ctype,
43              clkdivnum => clkdivnum -- How much clock division. Put to 1 to disable.
44          )
45          port map(
46              iControllerEnable      => iGraintopEnable, -- Enable signal from top. Paused if zero.
47              oControllerYFlag       => graintopYFlag, -- If keystream is available.
48              oControllerYEnable     => graintopYEnable, -- Input to F and G, is in init stage if 1. coming from
49  controller.
50              oControllerMsgNewval   => graintopMsgNewval, -- Input to accumlogic from controller, (Will be
51  MsgIn normally, but is modified during loading to save hardware)
52              iControllerMsgIn       => iGraintopMsg, -- Input to controller from Top.
53              oControllerAccumStart  => graintopAccumStart, -- (Decides whether or not accum should be used or
54  not. Will be Enable or 0 when y bit is keystream)
55              oControllerGrainStart  => grainTopGrainStart, -- Controls if the grain runs. Is automatically stopped
56  during accum
```

```
1           oControllerFP1        => grainTopFP1, -- Controls if the grain runs. Is automatically stopped during
2   accum
3           iControllerYFeedback   => grainTopYFeedback, -- Get the Y values used for accum
4           oControllerYAccum      => grainTopYAccum, -- Sends y values to accumulator
5           oControllerYOut       => grainTopYOut, -- Y values sent out (keystream.
6           iControllerCryptMode   => iGraintopCryptMode, -- 1 = decryption. 0 = Encryption
7           iControllerCryptEnable => iGraintopCryptEnable, -- Enable/disable encryption
8           iControllerReset       => iGraintopReset,
9           iControllerClk        => designClk,
10          iControllerClk2        => iGraintopClk -- For 1 para.
11      );
12
13
14
15
16      accumulatorInst : accumulator
17        generic map(
18          n1 => ndiv2, -- Parallellization/unrolling for accumulator. (half of the grain)
19          n2 => n -- Parallellization/unrolling for grain. n1*2
20        )
21        port map(
22          iAccumY          => graintopYAccumReg, -- Either first 15 or last 32 bits of Y (during loading) or every
23  second bit
24          iAccumReset      => iGraintopReset,
25          iAccumClk        => iGraintopClk,
26          iAccumStart      => graintopAccumStartReg,
27          oAccumRegFeedback => graintopAccumlogicRegFeedback,
28
29          iAccumMsg        => grainTopMsgReg
30        );
31
32      AuthPipeInst : authPipeline
33        generic map(
34          n     => n,
35          ndiv2 => ndiv2,
36          ia    => ia
37        )
38        port map(
39          iAuthPipelineMsgIn        => graintopMsgNewval,
40          oAuthPipelineMsgReg       => grainTopMsgReg,
41          iAuthPipelineYAccum       => grainTopYAccum,
42          oAuthPipelineYAccumReg    => grainTopYAccumReg,
43          iAuthPipelineAccumStart   => graintopAccumStart,
44          oAuthPipelineAccumStartReg => grainTopAccumStartReg,
45          iAuthPipelineClk          => iGraintopClk,
46          iAuthPipelineReset        => iGraintopReset
47        );
48
49      oGraintopY     <= grainTopYOut; -- keystream.
50      oGraintopYFlag <= graintopYFlag;
51      oGraintopTag   <= graintopAccumlogicRegFeedback; -- TagFlag will be high when we got the tag.
52      --------------------------------------------------------------------------------
53
54  end structural;
55
```

Τελευταίο παρουσιάζεται το περιεχόμενο του αρχείο GrainTB.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
---------------------------------
entity cryptoTb is
   -- Port ();
end cryptoTb;
-------------------------------------
architecture Behavioral of cryptoTb is
   -------------------------------------
   component grainTop is
      generic (
         n        : integer;
         ndiv2    : integer;
         g        : integer;
         yt       : integer;
         ctype    : integer;
         clkdivnum : integer;
         ia       : integer -- Isolation or not
      );
      port (
         oGraintopY          : out std_logic_vector(0 to (ndiv2) - 1); -- Key stream out.
         iGraintopIv         : in std_logic_vector(0 to n - 1); -- IV in.
         iGraintopKey        : in std_logic_vector(0 to n - 1); -- Key in
         iGraintopEnable     : in std_logic; -- Start
         iGraintopReset      : in std_logic;
         iGraintopClk        : in std_logic;
         oGraintopYFlag      : out std_logic; -- When keystream available.
         oGraintopTag        : out std_logic_vector(0 to 63); -- The tag if using authentication.
         iGraintopCryptMode  : std_logic; -- 1 = decryption. 0 = Encryption
         iGraintopCryptEnable : std_logic; -- Enable/disable encryption
         iGraintopMsg        : in std_logic_vector(0 to (ndiv2) - 1) -- Message
      );
   end component;
   ----------------------------------
   constant period               : time := 5 ns;
   constant halfperiod           : time := 2.5 ns;
   constant n                    : integer := 1;
   constant ndiv2                : integer := 1; -- Half of n in normal cases. If n is 1, this will be one as well.

   signal tbCryptMode            : std_logic := '1'; -- 0 = encryption. 1 = decryption.
   signal cryptMode              : integer := 1; -- Change this value with tbCryptmode. as the same value.
   signal tbCryptEnable          : std_logic := '1'; -- Enable/disable encryption/decryption.
   constant IndexStart           : integer := 0; -- When to encrypt start. Length of Adlen + AD. Put to like 9999
when there is no encryption.
   constant IndexEnd             : integer := 100; -- Everything but the last 8 bits.




   signal tbReset                : std_logic := '0';
```

```
    signal tbClk                : std_logic := '0';
    signal tbEnable             : std_logic := '0';
    signal tbMessage            : std_logic_vector(0 to ndiv2 - 1) := (others => '0');
    signal tbY                  : std_logic_vector(0 to ndiv2 - 1) := (others => '0');
    signal tbIv                 : std_logic_vector(0 to n - 1) := (others => '0');
    signal tbKey                : std_logic_vector(0 to n - 1) := (others => '0');
    signal tbYFlag              : std_logic := '0';

    signal tbTag                : std_logic_vector(63 downto 0); -- The tag is defined in the opposite way
according to the Grain-128a/grain-128AEAD paper.

    signal preout23             : std_logic_vector(0 to 319);
    signal preout1, preout2     : std_logic_vector(0 to 319);
    signal preout3, preout4     : std_logic_vector(0 to 255);
    signal key1, key2, key3, key4   : std_logic_vector(0 to 127);
    signal iv1, iv2, iv3, iv4       : std_logic_vector(0 to 127);
    signal accum1, reg1, accum2, reg2 : std_logic_vector(0 to 31);
    ---new added message---
    signal m0 : std_logic_vector(0 to 8);
    signal m1 : std_logic_vector(0 to 1);
    signal m2 : std_logic_vector(0 to 1);
    signal m3 : std_logic_vector(0 to 20); -- 20. 11 extra zeros
    signal m4 : std_logic_vector(0 to 41); -- 41. 6 extra zeros
    ----------------------------------

    --------------------------------------------------------------------------------
begin
    --------------------------------------------------------------------------------
    grainTopInst3 : grainTop
        generic map(-- Change when selecting optimization and controller.
        n       => n,
        ndiv2   => ndiv2, -- Put at 1 when doing 1 para. n/2
        g       => 0,
        yt      => 0,
        ctype   => 1,
        clkdivnum => 64, -- Put it to 2 by default so the program doesn't complain. Won't affect anything unless
ctype is 1.
        ia      => 1-- Isolation or not
        )
        port map(
            oGraintopY          => tbY, --these are wire or signal

            iGraintopIv         => tbIv,
            iGraintopKey        => tbKey,
            iGraintopEnable     => tbEnable,
            iGraintopReset      => tbReset,
            iGraintopClk        => tbClk,

            oGraintopTag        => tbTag, -- The tag if using authentication

            oGraintopYFlag      => tbYFlag,
            iGraintopMsg        => tbMessage,
            iGraintopCryptMode  => tbCryptMode, -- 1 = decryption. 0 = Encryption
            iGraintopCryptEnable => tbCryptEnable -- Enable/disable encryption

        );
```

```
        -------------------------

        tbClk <= not(tbClk) after halfperiod * 1;

        ---------------------------------------------------------------------------------------
        --test1

        key1 <= X"00000000000000000000000000000000";
        iv1  <= X"000000000000000000000000ffffffe";
        iv1 <= X"00000000000000000000000100000000";
        preout1 <=
X"c0207f221660650b6a952ae26586136fa0904140c8621cfe8660c0dec0969e9436f4ace92cf1ebb7";

        ---test2
        key2 <= X"0123456789abcdef123456789abcdef0";
        key2 <= (others=>'0');
        iv2     <= X"0123456789abcdef12345678ffffffe"; -- 80c4a2e691d5b3f7482c6a1e

        preout23 <=
X"c0207f221660650b6a952ae26586136fa0904140c8621cfe8660c0dec0969e9436f4ace92cf1ebb7";
        preout2  <=
X"f88720c13f46e6a43c07eeed89161a4dd73bd6b8be8b6b116879714ebb630e0a4c12f0399412982c";
        ----test3
        key3    <= (others => '0');
        iv3    <= X"800000000000000000000000ffffffe";
        accum1  <= X"564b3622"; --new add
        reg1    <= X"19bd90e3"; --new add
        preout3 <= X"01f259cf52bf5da9deb1845be6993abd2d3c77c4acb90e422640fbd6e8ae642a";


        ----test4
        key4    <= X"0123456789abcdef123456789abcdef0";
        iv4    <= X"8123456789abcdef12345678ffffffe";
        accum2  <= X"7f2acdb7"; --new add
        reg2    <= X"adfb701f"; --new add
        preout4 <= X"8d2083b3c32b43f1962b3dcabf679378db3536bfc25bed483008e6bcb395a156";

        ------5 different Message
        m0 <= "100000000";
        m1 <= "01";
        m2 <= "11";
        m3 <= "000100100011010000001";

        m4 <= "00010010001101000101011001111000100111101l";

        process
        variable MsgIndex  : integer := 0;
        variable MsgFinish : integer := 0;
    begin
        wait for halfperiod;

        for t in 0 to 3 loop
            for m in 0 to 2 loop
                MsgIndex  := 0; -- Set it 0 zero at the start.
                MsgFinish := 0;
                tbReset  <= '1';
```

```
1               tbEnable <= '0';
2               wait for 3 * period;
3               tbReset  <= '0';
4               tbEnable <= '1';
5               for f in 0 to 1 loop
6                  for k in 0 to (128/n) - 1 loop
7                     if (t = 0) then
8                        tbKey <= key1((k * n) to ((k + 1) * n - 1)); -- Input last bit first.
9                        tbIv  <= iv1((k * n) to ((k + 1) * n - 1));
10                    elsif (t = 1) then
11                       tbKey <= key2((k * n) to ((k + 1) * n - 1)); -- Input last bit first.
12                       tbIv  <= iv2((k * n) to ((k + 1) * n - 1));
13                    elsif (t = 2) then
14                       tbKey <= key3((k * n) to ((k + 1) * n - 1)); -- Input last bit first.
15                       tbIv  <= iv3((k * n) to ((k + 1) * n - 1));
16                    else
17                       tbKey <= key4((k * n) to ((k + 1) * n - 1)); -- Input last bit first.
18                       tbIv  <= iv4((k * n) to ((k + 1) * n - 1));
19                    end if;
20                    if (f = 1) then -- When doing accumload
21                       tbMessage <= (others => '0');
22                       if (k = 128/(2 * n)) then -- Half the accumload
23                          tbMessage(0) <= '1'; --ndiv2 -1
24                       end if;
25                    end if;
26                    wait for period;
27                 end loop;
28
29                 if (f = 0) then
30                    tbKey <= (others => '0'); -- Stop sending in key and IV
31                    tbIv  <= (others => '0'); -- Stop sending in
32                    wait until tbYFlag = '1';
33                 end if;
34
35                 if (f = 1) then
36                    tbKey <= (others => '0'); -- Stop sending in
37                    tbIv  <= (others => '0'); -- Stop sending in
38                 end if;
39              end loop;
40
41              tbCryptEnable <= '1';
42              case m is
43                 when 0 =>
44
45                    if (MsgFinish = 0) then
46
47                       tbMessage <= (others => '0'); -- Reset it.
48
49                       for g in 0 to m0'LENGTH - 1 loop -- (ndiv2 - 1) --(ndiv2 - 1)
50                          tbMessage((g mod (ndiv2))) <= m0(g); -- Sends in most signficant value first.
51                          --report integer'image(g) & ".Value. message = " & std_logic'image(m0(g)) &
52    integer'image((g MOD (n/2)));
53                          if ((g mod (ndiv2)) = ndiv2 - 1) then -- n/2 bit msg generated send it in and wait 1 clock
54    cycle.
55                             --report "New message";
56
```

```
1                           wait for period;
2                           tbMessage <= (others => '0'); -- Sets a default value for next msg round.
3                           if (n = 1) then -- In 1 para we need to wait 1 extra clock cycle since the next msg round
4   won't be until the next accum/mac bit is generated
5                               wait for period; -- One extra wait.
6                           end if;
7                           --tbMessage <= (others => '0'); -- Sets a default value for next msg round.
8                       elsif (g = m0'LENGTH - 1) then -- Message is smaller or not dividable by parallelization.
9   Pads on zeros
10                          wait for period;
11                          tbMessage <= (others => '0'); -- Sets a default value for next msg round.
12
13                      end if;
14
15                  end loop;
16                  MsgFinish := 1; -- message finish, we stop encrypting/decrypting
17
18                  wait for period;
19                  tbCryptEnable <= '0'; -- we stop encrypting/decryption when the message is sent in.
20                  tbMessage     <= (others => '0');
21
22              else -- In all clock cycles after message finished.
23                  tbMessage <= (others => '0');
24                  -- tbCryptEnable <= '0'; -- we stop encrypting/decryption when the message is sent in.
25              end if;
26          when 1 =>
27
28              if (MsgFinish = 0) then
29                  tbMessage <= (others => '0'); -- Reset it.
30                  for g in 0 to m1'LENGTH - 1 loop --(ndiv2) - 1) -
31                      tbMessage(((g mod (ndiv2)))) <= m1(g); -- Sends in most signficant value first.
32
33                      if ((g mod (ndiv2)) = ndiv2 - 1) then -- n/2 bit msg generated send it in and wait 1 clock
34  cycle.
35
36                          wait for period;
37                          tbMessage <= (others => '0'); -- Sets a default value for next msg round.
38                          if (n = 1) then -- In 1 para we need to wait 1 extra clock cycle since the next msg round
39  won't be until the next accum/mac bit is generated
40                              wait for period; -- One extra wait.
41                          end if;
42                          --tbMessage <= (others => '0'); -- Sets a default value for next msg round.
43                      elsif (g = m1'LENGTH - 1) then -- Message is smaller or not dividable by parallelization.
44  Pads on zeros
45                          wait for period;
46                          tbMessage <= (others => '0'); -- Sets a default value for next msg round.
47
48                      end if;
49
50                  end loop;
51                  MsgFinish := 1;
52                  tbCryptEnable <= '0'; -- we stop encrypting/decryption when the message is sent in.
53                  wait for period;
54                  tbCryptEnable <= '0'; -- we stop encrypting/decryption when the message is sent in.
55                  tbMessage     <= (others => '0');
56              else
```

```
                    tbMessage <= (others => '0');
                    -- tbCryptEnable <= '0'; -- we stop encrypting/decryption when the message is sent in.
                end if;
            when 2 =>

                if (MsgFinish = 0) then
                    tbMessage <= (others => '0'); -- Reset it.
                    for g in 0 to m3'LENGTH - 1 loop-- ((ndiv2) - 1) -
                        tbMessage((g mod (ndiv2))) <= m3(g); -- Sends in most signficant value first.

                        if ((g mod (ndiv2)) = (ndiv2) - 1) then -- n/2 bit msg generated send it in and wait 1 clock
cycle.

                            wait for period;
                            tbMessage <= (others => '0'); -- Sets a default value for next msg round.
                            if (n = 1) then -- In 1 para we need to wait 1 extra clock cycle since the next msg round
won't be until the next accum/mac bit is generated
                                wait for period; -- One extra wait.
                            end if;
                            --tbMessage <= (others => '0'); -- Sets a default value for next msg round.
                        elsif (g = m3'LENGTH - 1) then -- Message is smaller or not dividable by parallelization.
Pads on zeros
                            wait for period;
                            tbMessage <= (others => '0'); -- Sets a default value for next msg round.

                        end if;

                    end loop;
                    MsgFinish := 1;

                    wait for period;
                    tbCryptEnable <= '0'; -- we stop encrypting/decryption when the message is sent in.
                    tbMessage     <= (others => '0');
                else
                    tbMessage <= (others => '0');
                    --tbCryptEnable <= '0'; -- we stop encrypting/decryption when the message is sent in.
                end if;

        end case;


        wait for period;
        --end loop;

        -- Here we are done!

        wait for 5 * period;

    end loop;

  end loop;
  wait for halfperiod; -- For starting over.


end process;
```

```
1       ----------------------------------------------------------------------------
2
3   end Behavioral;
4
5
```