Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή
Στην Ασφάλεια Υπολογιστών και Δικτύων

Μελέτη και αξιολόγηση αλγορίθμων μετακβαντικής κρυπτογραφίας

Αντώνιος Αλεξίου

Επιβλέπων Καθηγητής
Κωνσταντίνος Λιμνιώτης

Δεκέμβριος 2018

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μελέτη και αξιολόγηση αλγορίθμων μετακβαντικής κρυπτογραφίας

Αντώνιος Αλεξίου

Επιβλέπων Καθηγητής
Κωνσταντίνος Λιμνιώτης

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

μεταπτυχιακού τίτλου σπουδών
στην Ασφάλεια Υπολογιστών και Δικτύων

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Δεκέμβριος 2018

Περίληψη

Η μελέτη κρυπτογραφικών αλγορίθμων δημοσίου κλειδιού, οι οποίοι θα είναι ανθεκτικοί και μετά την έλευση των κβαντικών υπολογιστών, αποτελούν ένα πολύ σημαντικό ερευνητικό πεδίο, καθώς οι σημερινοί κλασικοί αλγόριθμοι δημοσίου κλειδιού (RSA, ελλειπτικών καμπυλών κ.α.) δεν θα παρέχουν πλέον καμία ασφάλεια στο νέο αυτό τεχνολογικό περιβάλλον. Ήδη γνωρίζουμε διάφορους κρυπτογραφικούς αλγορίθμους οι οποίοι είναι, ή φέρονται να είναι, ασφαλείς ακόμα και με την υπόθεση ότι διαθέτουμε κβαντικούς υπολογιστές (ήτοι πρόκειται για αλγόριθμους μετακβαντικής κρυπτογραφίας). Ωστόσο πολλά θέματα ως προς τη δυνατότητα υλοποίησής τους παραμένουν ανοιχτά.

Στην παρούσα εργασία μελετάται η μετακβαντική κρυπτογραφία και τα συναφή μαθηματικά προβλήματα πάνω στα οποία δομούνται οι μετακβαντικοί αλγόριθμοι, με έμφαση στα προβλήματα που άπτονται της θεωρίας κωδίκων, παρουσιάζοντας και υλοποιώντας το κρυπτοσύστημα McEliece – ένα κλασικό κρυπτογραφικό σύστημα που αποτελεί ήδη έναν από τους υποψήφιους κρυπταλγορίθμους στη διαδικασία προτυποποίησης που έχει εκκινήσει ο οργανισμός NIST. Απώτερος στόχος είναι η συγκριτική αποτίμηση της απόδοσης του McEliece σε εφαρμογές πραγματικού χρόνου, σε σχέση με σημερινούς συμβατικούς κρυπταλγορίθμους. Αρχικά πραγματοποιείται μία βιβλιογραφική επισκόπηση σε θεωρητικά ζητήματα της κλασσικής και μετακβαντικής κρυπτογραφίας αλλά και σε θέματα της θεωρίας της πληροφορίας με έμφαση στον κβαντικό υπολογιστή. Στη συνέχεια παρουσιάζονται τα χαρακτηριστικά του κρυπτοσυστήματος McEliece. Αναπτύσσεται μια υλοποίηση κρυπτοσυστήματος δημόσιου κλειδιού RSA, που αποτελεί και έναν κλασσικό αλγόριθμο, καθώς επίσης και μία αντίστοιχη υλοποίηση κρυπτοσυστήματος δημοσίου κλειδιού με τον αλγόριθμο McEliece: και οι δύο υλοποιήσεις αφορούν μία τυπική εφαρμογή συνομιλίας (chat). Ακολούθως, γίνεται συγκριτική αποτίμηση των δύο υλοποιήσεων. Η μελέτη καταδεικνύει ότι πράγματι ο αλγόριθμος McEliece, ως αλγόριθμος μετακβαντικής κρυπτογραφίας, είναι σημαντικά πιο αργός, αναδεικνύοντας την ανάγκη εύρεσης μεθόδων βελτίωσης της απόδοσης των αλγορίθμων αυτής της κατηγορίας.

Λέξεις-κλειδιά:

McEliece, Postquantum Cryptography, Public Key, RSA, Goppa, chat, quantum-proof, java, Quantum Computer.

Summary

The study of public key cryptographic algorithms, which will be resistant after the arrival of quantum computers, constitutes a very important research field, given the fact that today's classical public key algorithms (RSA, elliptic curves etc.) will no longer provide any security in this new technological environment. We already know a variety of cryptographic algorithms that are, or are supposed to be, secure, even under the hypothesis that quantum computers are a reality (that is, post-quantum cryptographic algorithms). However, many aspects of their efficiency remain open.

In this thesis, post-quantum cryptography is being studied, analyzing the mathematical problems which provide the means for constructing post-quantum cryptographic algorithms. Emphasis is given on problems based on coding theory and especially on the McEliece cryptosystem, which is a typical postquantum cryptographic algorithm that has been already submitted to the relevant ongoing NIST competition for standardization. Our ultimate goal is a comparative study between McEliece and classical modern ciphers in terms of efficiency, when they are to be used in real time applications. First a bibliographic review on the theoretical issues of classical and post-quantum cryptography, as well as on information theory, is presented, putting also emphasis on the quantum computer. Next, we present the main characteristics of the McEliece cryptosystem. In the framework of this thesis, we developed an RSA public key cryptosystem implementation, which is a massively used classic algorithm, as well as an implementation of a public key cryptosystem using the McEliece algorithm: both implementations have been incorporated in a chat application. A comparative assessment of these two implementations has been held, illustrating that the McEliece cipher is by far less efficient and, thus, we conclude that much effort should be put to further improve the performance of such post-quantum ciphers.

Keywords:

McEliece, Postquantum Cryptography, Public Key, RSA, Goppa, chat, quantum-proof, java, Quantum Computer.

Ευχαριστίες

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα την μεταπτυχιακή μου διατριβή, καθηγητή Κωνσταντίνο Λιμνιώτη, για την καθοδήγηση αλλά και την επιστημονική του συμβολή στην σωστή μελέτη και εκπόνηση της παρούσης διατριβής.

Ευχαριστώ δε, θερμά την οικογένειά μου, ήτοι τη σύζυγο μου Εύα, αλλά και τον μικρό Γιάννη για την υποστήριξη και διευκόλυνση που μου παρείχαν καθόλη τη διάρκεια της μελέτης.

Περιεχόμενα

Κεφάλαιο 1
Δομή Διατριβής

1.1 Η κρυπτογραφία σήμερα

Η κρυπτογραφία είναι εκείνη η επιστήμη που μελετά εφαρμοσμένες τεχνικές με τις οποίες ένα μήνυμα μετασχηματίζεται σε κρυπτοκείμενο κατά τέτοιον τρόπο ώστε να μην μπορεί να γίνει αντιληπτό το περιεχόμενο του από κάποιον που καταφέρνει να το λάβει. Η διαδικασία αυτή ονομάζεται κρυπτογράφηση. Αντιστροφή αποτελεί η διαδικασία της αποκρυπτογράφησης. Στις μέρες μας η κρυπτογραφία έχει ευρύτερη έννοια και αναφέρεται στην εφαρμογή μαθηματικών τεχνικών ώστε να αντιμετωπιστούν ζητήματα ασφάλειας της πληροφορίας και ειδικότερα θέματα:

i. Εμπιστευτικότητα(Confidentiality)

ii. Πιστοποίηση ταυτότητας του αποστολέα (Authentication)

iii. Διασφάλιση του αδιάβλητου (ακεραιότητας) της πληροφορίας (Integrity)

Οι σύγχρονες εφαρμογές της κρυπτογραφίας αφορούν, μεταξύ άλλων, την ασφάλεια στο Διαδίκτυο (π.χ. η περίπτωση του πρωτοκόλλου https) καθώς ο χρήστης πλοηγείται μέσα σε αυτό το περιβάλλον. Η κρυπτογραφία στο Διαδίκτυο συμπεριλαμβάνει και την ασφάλεια του ηλεκτρονικού ταχυδρομείου, των τραπεζικών συναλλαγών, των εικονικών ιδιωτικών δικτύων (VPN), αλλά ακόμη και τη μετάδοση φωνής πάνω από διαδίκτυο (VoIP). Η πρόσβαση στο Διαδίκτυο με τη χρήση ασύρματων δικτυών αποτελεί ένα ακόμη ευρύ πεδίο εφαρμογής της κρυπτογραφίας. Οι καθημερινές μας συνομιλίες μέσω δικτύων κινητής τηλεφωνίας εξασφαλίζονται ως απόρρητες με κρυπτογραφικά πρωτόκολλα.

Πέραν των επικοινωνιών, εταιρείες μεγάλου βεληνεκούς έχουν τεράστιες βάσεις και δεδομένα τα οποία πρέπει να διατηρούνται ασφαλή και η πρόσβαση να ελέγχεται, κάτι που διασφαλίζεται με κρυπτογραφικές μεθόδους. Ιδιαίτερη έμφαση δίδεται σε «ευαίσθητα» δίκτυα όπως εκείνα της διπλωματίας και του στρατού αλλά και κάποια αλλά δίκτυα ad-hoc . Επίσης, το κρυπτονόμισμα και η ύπαρξή του βασίζεται στην κρυπτογραφία και σε ειδικές τεχνικές της. Όλα τα ανωτέρω αποτελούν κάποια ενδεικτικά παραδείγματα στα οποία βρίσκουν άμεση εφαρμογή σήμερα οι κρυπτογραφικοί αλγόριθμοι. Η ασφάλεια των σημερινών κρυπτογραφικών αλγορίθμων είναι θεμελιωμένη ιδίως σε συγκεκριμένες μαθηματικές ιδιότητες.

1.2 Η κρυπτογραφία στο μέλλον

Με την έλευση του κβαντικού υπολογιστή τις επόμενες δεκαετίες, πολλοί ευρέως διαδεδομένοι αλγόριθμοι κρυπτογράφησης απειλούνται. Για αυτό το λόγο, η επιστημονική κοινότητα προετοιμάζεται πάνω σε πλαίσια κρυπτογραφικών αλγορίθμων οι οποίοι, θεωρητικά, δεν προσκρούουν στις δυνατότητες του κβαντικού υπολογιστή. Η αντιμετώπιση του ζητήματος της εύρεσης και χρήσης μαθηματικών προβλημάτων, τα οποία δεν λύνονται σε πολυωνυμικό χρόνο από κβαντικό υπολογιστή, ονομάζεται επιστήμη της μετακβαντικής κρυπτογραφίας [10]. Έχει ως στόχο να εξασφαλίσει την συνέχεια (continuity) των εργασιών στο διαδίκτυο και σε όποια εφαρμογή θα χρειαστεί να περάσει(μετακβαντικά) σε άλλο κρυπτογραφικό πρωτόκολλο, έτσι ώστε να διασφαλίσει τα κεκτημένα της κρυπτογραφίας όπως αυτή χρησιμοποιείται σήμερα. Αναφέρθηκαν πολλοί τομείς χρήσης στην καθημερινότητα, ποικίλων κρυπτογραφικών τεχνικών. Εκείνη η περιοχή που πλήττεται περισσότερο υπό τον κίνδυνο του κβαντικού υπολογιστή είναι κυρίως τα κρυπτοσυστήματα δημοσίου κλειδιού και ψηφιακών υπογραφών. Με άλλα λόγια, σημερινοί κρυπτογραφικοί αλγόριθμοι της κατηγορίας δημοσίου κλειδιού, καθώς και

κρυπτογραφικά σχήματα ψηφιακών υπογραφών, δεν θα παρέχουν καμία ασφάλεια με την έλευση των κβαντικών υπολογιστών – άρα και θα πρέπει, τελικά, να αντικατασταθούν. Σε αυτά εστιάζει η ακαδημαϊκή κοινότητα των μαθηματικών, των κρυπτογράφων και των κρυπταναλυτών, ώστε να ευρεθούν αλγόριθμοι τέτοιοι, που να αντέχουν στην κρυπτανάλυση με κβαντικό υπολογιστή. Ήδη ο οργανισμός προτυποποίησης NIST έχει εκκινήσει διαδικασία για την επιλογή πρότυπων αλγορίθμων μετακβαντικής κρυπτογραφίας, στο πλαίσιο της οποίας έχει προταθεί πλήθος αλγορίθμων που είναι ήδη υπό μελέτη από την επιστημονική κοινότητα. Αυτοί οι αλγόριθμοι εστιάζουν σε γνωστά δύσκολα μαθηματικά προβλήματα, η δυσκολία των οποίων δεν επηρεάζεται ούτε από τους κβαντικούς υπολογιστές, όπως θα δούμε σε επόμενο κεφάλαιο.

1.3 Ερευνητικά ερωτήματα

Η παρούσα εργασία εστιάζει στη μετακβαντική κρυπτογραφία, πραγματοποιώντας μία επισκόπηση του χώρου και εξετάζοντας θέματα απόδοσης αλγορίθμων μετακβαντικής κρυπτογραφίας, σε σχέση με «ισοδύναμους» χρησιμοποιούμενους αλγόριθμους. Το κύριο ερώτημα στο οποίο εστιάζει η παρούσα διατριβή είναι το πόσο αποδοτικοί, ως προς την ταχύτητα, είναι οι αλγόριθμοι μετακβαντικής κρυπτογραφίας σε σχέση με τους συμβατικούς σημερινούς αλγορίθμους, σε εφαρμογές που απαιτούν επικοινωνίες σε πραγματικό χρόνο. Το ερώτημα αυτό έχει ιδιάζουσα σημασία γιατί, παρόλο που γνωρίζουμε ότι κατά κανόνα οι αλγόριθμοι μετακβαντικής κρυπτογραφίας είναι λιγότερο αποδοτικοί, εν τούτοις διαφαίνεται ότι θα πρέπει να αρχίσουν να υλοποιούνται και να εφαρμόζονται και στους συμβατικούς (μη κβαντικούς) σημερινούς υπολογιστές, προκειμένου να «προετοιμαστεί» το έδαφος για τη μετάβαση στην κβαντική εποχή.

Ειδικότερα, στην παρούσα εργασία εξετάζεται η απόδοση ενός συγκεκριμένου μετακβαντικού κρυπτογραφικού αλγορίθμου δημοσίου κλειδιού. Ο αλγόριθμος αυτός είναι ο McEliece, η απόδοση του οποίου συγκρίνεται με αυτή του γνωστού σημερινού αλγορίθμου RSA. Επιλέξαμε αυτό τον αλγόριθμο διότι είναι εδραιωμένος πολλά χρόνια στην ακαδημαϊκή κοινότητα και επίσης έχει υποβληθεί στον 1ο γύρο προτάσεων στο NIST[21],[22]. Για την αξιολόγηση της απόδοσης τους αλγορίθμου, εστιάσαμε σε μία εφαρμογή συνομιλίας (chat) και στις ακόλουθες κρυπτογραφικές λειτουργίες, οι οποίες συναντώνται κατά κανόνα σε κάθε κρυπτογραφημένη επικοινωνία που απαιτείται και αυθεντικοποίηση των δύο μελών:

i. δημιουργία ζεύγους δημοσίου και ιδιωτικού κλειδιού.

ii. αποστολή δημοσίου κλειδιού.

iii. αποστολή συμμετρικού κλειδιού.

1.4 Αντικείμενο της διατριβής

Σε πειραματικό περιβάλλον δυο συστημάτων υπολογιστών με χαρακτηριστικά που αναφέρονται στο αντίστοιχο κεφάλαιο, υλοποιήθηκε μια εφαρμογή όπως παρουσιάζεται στη συνέχεια. Έχοντας μελετήσει το χώρο της μετακβαντικής κρυπτογραφίας και ειδικότερα κάποιους ενδεικτικούς αλγόριθμους της κατηγορίας αυτής, γίνεται η ανάπτυξη κρυπτογραφικής εφαρμογής στην οποία ο κλασικός αλγόριθμος δημοσίου κλειδιού αντικαθίσταται από αλγόριθμο μετακβαντικής κρυπτογραφίας (τον αλγόριθμο McEliece), προκειμένου να διερευνηθεί το κατά πόσον επηρεάζεται η απόδοση (ταχύτητα) των εφαρμογών αυτών και να υπάρξει έτσι μία συγκριτική αποτίμηση. Στη συγκεκριμένη περίπτωση, αναπτύσσεται και μελετάται μια εφαρμογή συνομιλίας με υλοποίηση τόσο με αλγόριθμο κλασσικής κρυπτογραφίας όσο και μετακβαντικής κρυπτογραφίας σε διάφορα στάδια της επικοινωνίας, προκειμένου να διαφανεί η απόδοσή τους σε εφαρμογές «πραγματικού χρόνου», πραγματοποιώντας συγκριτική αποτίμηση της απόδοσης της εφαρμογής με χρήση αλγορίθμου μετακβαντικής κρυπτογραφίας σε σχέση με την κλασική περίπτωση αλγορίθμου δημοσίου κλειδιού που χρησιμοποιείται σήμερα. Ειδικότερα, γίνεται χρήση των αλγορίθμων RSA2048/AES256 και η υλοποίηση ενός κρυπτοσυστήματος δημοσίου κλειδιού με τα standards αυτά. Το μέγεθος κλειδιού για τον RSA επιλέχθηκε να είναι στα 2048 bits καθώς αυτό αποτελεί ένα τυπικό μήκος κλειδιού σήμερα σε διάφορες εφαρμογές. Επιπλέον είναι ίδιου επιπέδου ασφάλειας με αυτό που παρέχει ο αλγόριθμος McEliece με μέγεθος κλειδιού στα 1024 bits [05]. Στη συνέχεια προχωρούμε στην υλοποίηση ενός κρυπτοσυστήματος McEliece1024/AES256 αντίστοιχα. Η υλοποίηση γίνεται σε περιβάλλον Windows Eclipse Photon σε Java. Κύριος στόχος είναι η εφαρμογή γνωστών βιβλιοθηκών ανοιχτού κώδικα αλλά και η δική μας ανάπτυξη λογισμικού, όπου χρειάζεται, με σκοπό τη δημιουργία ενός μοντέλου Client/Server chat και στα δυο προαναφερθέντα κρυπτοσυστήματα δημόσιου κλειδιού. Ειδικότερα, η εφαρμογή που αναπτύσσουμε περιλαμβάνει την:

i. Δημιουργία ζεύγους κλειδιών στον Client και στον Server [RSA 2048 /McEliece 1024].

ii. Ανταλλαγή δημοσίου κλειδιού.

iii. Δημιουργία συμμετρικού κλειδιού AES-256.

iv. χρήση RSA / McEliece για την κρυπτογράφηση του συμμετρικού AES.

v. Ασφαλή αποστολή του συμμετρικού κλειδιού.

Γενικότερα, η υλοποίηση περιλαμβάνει την χρήση sockets σε mode ανταλλαγής Bytes ή και objects, ενώ ενδεχόμενες πήγες ανοιχτών βιβλιοθηκών είναι οι Bouncy Castle και η Flexi-provider.

Ακολούθως, παρουσιάζονται αποτελέσματα μελέτης της απόδοσης αυτών των αλγορίθμων κυρίως όσον αφορά σε επίπεδο λειτουργικού συστήματος. Ακολουθεί η παρουσίαση των συγκριτικών χρόνων δημιουργίας και αποστολής δημοσίου αλλά και συμμετρικού κλειδιού με εστίαση στις διαφορές που θα προκύψουν από την μικρότερη πολυπλοκότητα του RSA σε σύγκριση με τον McEliece. Τα αποτελέσματα αποτυπώνονται για δυο βασικά λειτουργικά συστήματα. Ακολουθεί μια σύγκριση των τεχνικών που χρησιμοποιήθηκαν.

1.5 Δομή διατριβής

Η παρούσα εργασία ξεκινά με τη βιβλιογραφική αναφορά στην εξέλιξη των συστημάτων κρυπτογραφίας. Στο κεφάλαιο 2 κάνουμε μια σύντομη περιγραφή των συστημάτων συμμετρικού κλειδιού και τον αλγόριθμο AES.

Στο κεφάλαιο 3 παρουσιάζουμε τα σημαντικότερα μαθηματικά προβλήματα στα οποία βασίζονται τα συστήματα δημοσίου κλειδιού (public key cryptosystems - PKCS) [32]. Ειδικότερα, παρουσιάζουμε το πρόβλημα διακριτού λογαρίθμου και το ζήτημα της παραγοντοποίησης μεγάλων αριθμών. Εν συνεχεία, γίνεται ανάλυση σε δύο εκ των πιο διαδεδομένων κρυπτοσυστημάτων αυτής της κατηγορίας, στο πρωτόκολλο Diffie-Hellman και στον αλγόριθμο RSA. Επιπλέον, παρουσιάζουμε σύντομα την ασφάλεια στον RSA και ενδεικτικές κρυπταναλυτικές τεχνικές εναντίον του αλγορίθμου.

Στο 4ο κεφάλαιο μελετούμε και παρουσιάζουμε σημαντικά ερευνητικά θέματα της εξέλιξης της τεχνολογίας των υπολογιστών και δη, των κβαντικών υπολογιστών και τις βασικές αρχές λειτουργίας τους. Με την εμφάνιση της τεχνολογίας αυτής τίθενται σε αμφισβήτηση οι υπάρχουσες κρυπτογραφικές μέθοδοι οπότε και παρουσιάζεται ο αλγόριθμος Shor ο οποίος, όταν θα εκτελείται σε κβαντικούς υπολογιστές, μπορεί να απειλήσει και να καταρρίψει την ασφάλεια των σημερινών ευρέως χρησιμοποιούμενων αλγορίθμων δημοσίου κλειδιού αλλά και να απειλήσει αλγορίθμους συμμετρικού κλειδιού. Στο πλαίσιο αυτό, δίνεται έμφαση στην παρουσίαση της μηχανής Turing αλλά και των βασικών αρχών λειτουργίας του κβαντικού υπολογιστή.

Στο κεφάλαιο 5 μελετούμε και παρουσιάζουμε την ανάγκη χρήσης μετακβαντικών (quantum proof) κρυπταλγορίθμων γενικότερα [06],[08],[10] αλλά και ειδικότερα, σε προϊόντα που σήμερα κατασκευάζονται και έχουν μεγάλο προσδόκιμο ζωής. Επίσης αναλύουμε τα μαθηματικά προβλήματα που είναι υποψήφια για να αποτελέσουν τη βάση, προκειμένου να αναπτυχθούν οι μετακβαντικοί κρυπτογραφικοί αλγόριθμοι, με συγκριτική αποτίμησή τους. Η παρούσα διατριβή εστιάζει σε ένα θέμα με τρέχον ερευνητικό ενδιαφέρον, λαμβάνοντας υπόψη ότι ήδη ο οργανισμός προτυποποίησης NIST έχει εκκινήσει διαδικασία για την επιλογή πρότυπου αλγορίθμου μετακβαντικής κρυπτογραφίας.

Στο κεφάλαιο 6 μελετώνται βασικά στοιχεία της θεωρίας κωδίκων και ορίζονται τα προβλήματα της κλάσης πολυπλοκότητας NP-complete[10]. Έπειτα εστιάζουμε στους κώδικες τύπου Goppa που χρειάζονται στην υλοποίηση του κρυπτοσυστημάτων McEliece [15], [28].

Ακολούθως, στο κεφάλαιο 7 παρουσιάζεται το κρυπτοσύστημα McEliece το οποίο προτείνεται ως σύστημα μετακβαντικής κρυπτογραφίας. Εξετάζονται κάποια θέματα ασφάλειάς του, με τη χρήση κρυπταναλυτικών τεχνικών[03]. Μελετώνται προτάσεις που έχουν γίνει κατά καιρούς για κάποιες βελτιώσεις.

Στο κεφάλαιο 8 παρουσιάζεται η εφαρμογή που έχουμε υλοποιήσει στο πλαίσιο της παρούσας διατριβής [07]. Αναφέρονται κάποια σημαντικά στοιχεία του πηγαίου κώδικα και τέλος γίνεται παρουσίαση των μετρήσεων και μια συνολική αποτίμηση σε σχέση με τα ερευνητικά ερωτήματα που τέθηκαν.

Στο κεφάλαιο 9 παρουσιάζεται μια αποτύπωση της χρηστικότητας και μελλοντικής εφαρμογής σε πραγματικό περιβάλλον, υπολογιστικών συστημάτων και διαδικτύου., καταγράφοντας τα κύρια συμπεράσματα της διατριβής.

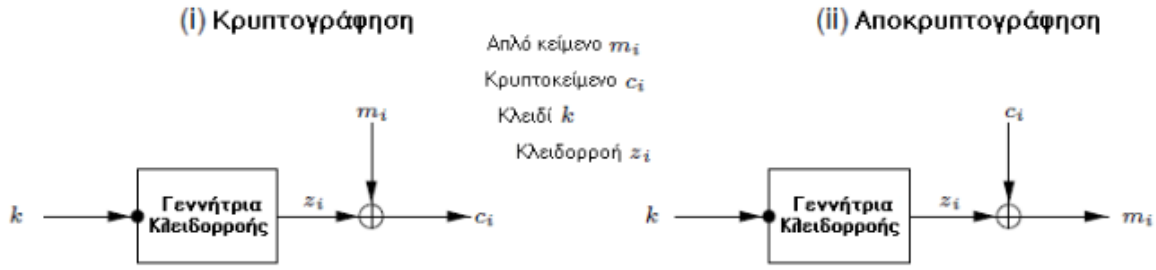
Κεφάλαιο 2
Κρυπτογραφία συμμετρικού κλειδιού

2.1 Συμμετρικό κλειδί

Με τον όρο κρυπτογραφία συμμετρικού κλειδιού αποδίδουμε την κρυπτογραφική προσέγγιση στην οποία ο αποστολέας και ο παραλήπτης – και μόνον αυτοί - γνωρίζουν ένα κοινό μυστικό κλειδί (symmetric-secret), όπου με το ίδιο κλειδί γίνεται τόσο η κρυπτογράφηση όσο και η αποκρυπτογράφηση του μηνύματος. Σήμερα, συναντούμε παντού τη χρήση συμμετρικού κλειδιού σε εφαρμογές, π.χ. στο Διαδίκτυο αλλά και σε συστήματα κινητής τηλεφωνίας. Πρότυπος αλγόριθμος κρυπτογράφησης συμμετρικού κλειδιού είναι ο AES [16].

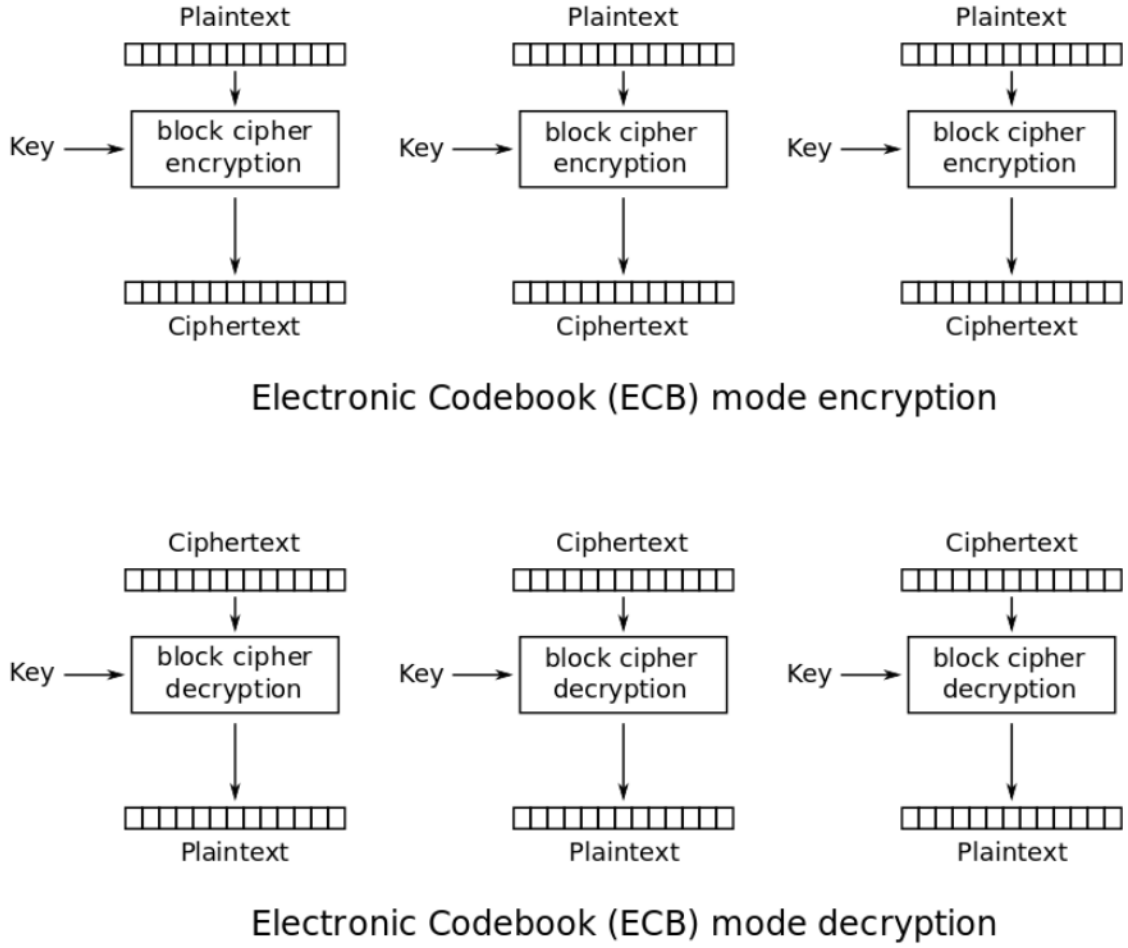
2.2 Είδη κρυπτογράφησης συμμετρικού κλειδιού

Οι αλγόριθμοι συμμετρικού κλειδιού διακρίνονται είτε σε κρυπταλγορίθμους ροής (stream ciphers), είτε σε κρυπταλγορίθμους τμήματος (block ciphers). Οι κρυπταλγόριθμοι ροής βρίσκουν ιδίως εφαρμογή σε περιπτώσεις όπου παραλήπτης και αποστολέας ανταλλάσσουν κρυπτογραφημένη πληροφορία και η ανάγκη για ταχύτητα είναι μεγάλη, ενώ υπάρχουν και περιορισμοί στην υπολογιστική ισχύ ή στην κατανάλωση ενέργειας των σχετικών συσκευών. Ένα τέτοιο παράδειγμα αποτελεί η κινητή τηλεφωνία με τον παλαιό κρυπταλγόριθμο A5/1[26]. Η όλη πολυπλοκότητα και ψευδοτυχαιότητα έγκειται στην υλοποίηση τέτοιων κρυπταλγορίθμων σε επίπεδο υλισμικού(hardware) αλλά και λογισμικού αναλόγως την εφαρμογή. Η κρυπτογράφηση και αποκρυπτογράφηση στους κρυπταλγορίθμους ροής γίνεται bit προς bit σε αντίθεση με τους κρυπταλγορίθμους τμήματος. Γνωστοί κρυπταλγόριθμοι ροής είναι ο RC4 (χρησιμοποιούνταν για χρόνια σε πολλές εφαρμογές, ενώ μέχρι πρόσφατα ήταν ο μόνος κρυπταλγόριθμος ροής στα διαδεδομένα πρωτόκολλα ασφαλείας SSL-TLS), ο ChaCha20 (ο οποίος αντικατέστησε τον RC4 στη νέα έκδοση του πρωτοκόλλου TLS) και ο E0 (bluetooth).



Εικόνα 2.1: Γενικό μοντέλο κρυπταλγορίθμου ροής [25].

Αντιθέτως, οι κρυπταλγόριθμοι τμήματος που είναι και οι πιο διαδεδομένοι, έχουν μοναδιαίο στοιχείο κρυπτογράφησης το λεγόμενο τμήμα (block) και όχι το bit (όπως στον κρυπταλγόριθμο ροής). Χρησιμοποιούνται στις καθημερινές διαδικτυακές δραστηριότητες μας, χωρίς να το γνωρίζουμε -στο https (http «πάνω» από το TLS), στα εικονικά ιδιωτικά δίκτυα (VPN), σε διαδικτυακές συναλλαγές Online καθώς και στο ηλεκτρονικό ταχυδρομείο. Το πρότυπο κρυπτογράφησης AES που αναφέρθηκε νωρίτερα είναι κρυπταλγόριθμος τμήματος.



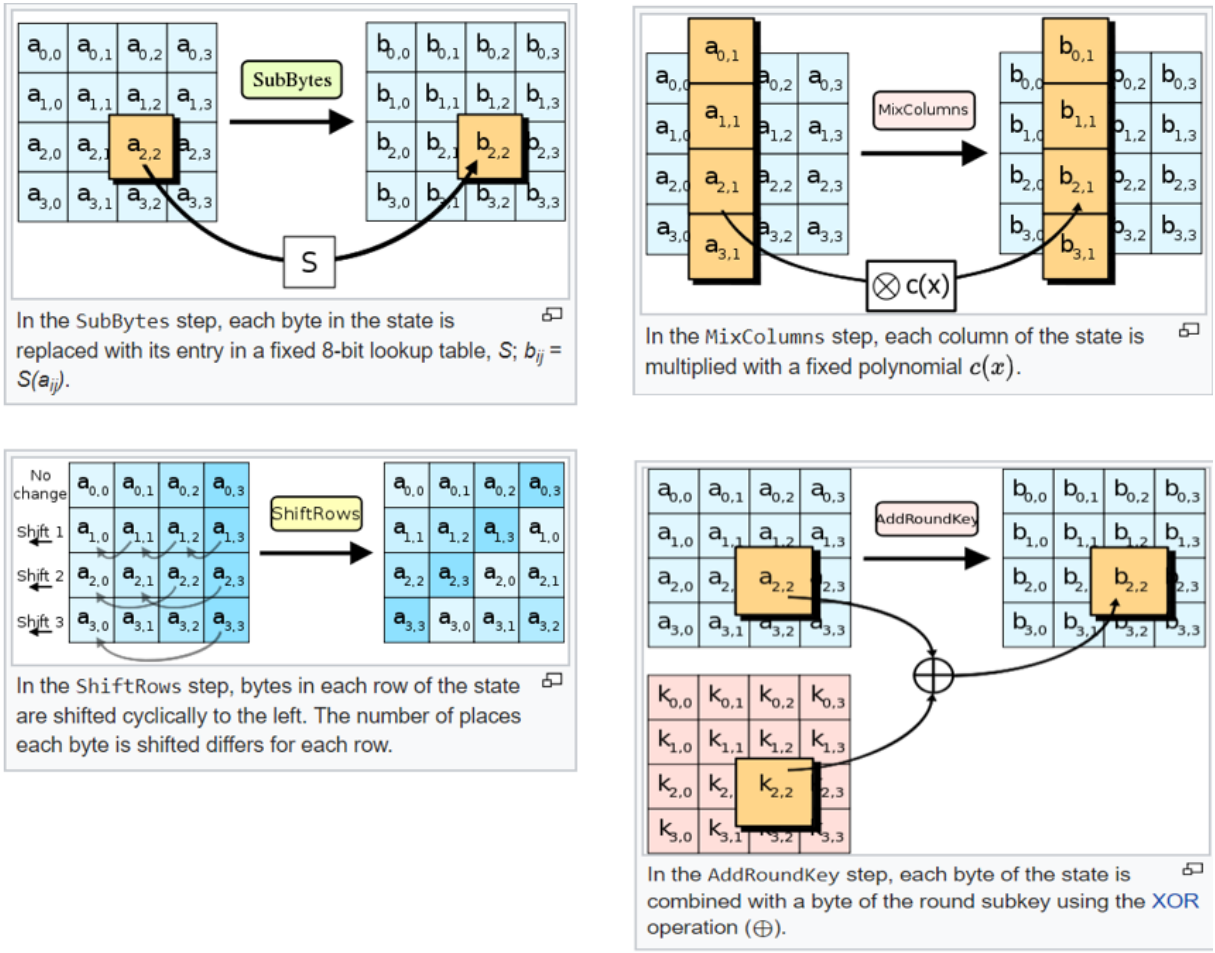
Εικόνα 2.2: Γενικό μοντέλο κρυπταλγορίθμου τμήματος(τρόπος λειτουργίας ECB) [30].

2.3 Ο AES και η εξέλιξή του

Ο αλγόριθμος αυτός προέκυψε μετά την αντικατάσταση του παλαιότερου προτύπου DES από τον οργανισμό NIST το 1997-1998. Μετά από αξιολόγηση προτεινομένων αλγορίθμων το 2000, ο αλγόριθμος Rijndael επελέγη ως ο νέος πρότυπος αλγόριθμος με το όνομα AES. Συνεπώς, το πρότυπο AES αναφέρεται μονοσήμαντα στη υλοποίηση και χρήση του Rijndael. Τα διάφορα μήκη κλειδιού που υποστηρίζει είναι 128, 192 & 256 bits και υλοποιείται με μέγεθος τμήματος (block size) στα 128bit.

Ο AES λαμβάνει χώρα σε 10-15 γύρους, με κάθε γύρο να περιλαμβάνει αντικατάσταση byte (Byte substitution) με s-boxes τα οποία έχουν αρκούντως καλά χαρακτηριστικά μη γραμμικότητας. Αμέσως μετά ακολουθεί η Ολίσθηση (Shift row) και ο συνδυασμός πολλών bit (Mix Column) .Τέλος έχουμε μια πρόσθεση (XOR) του κλειδιού. Όλη αυτή η διαδικασία συμβαίνει πάνω σε ένα πίνακα

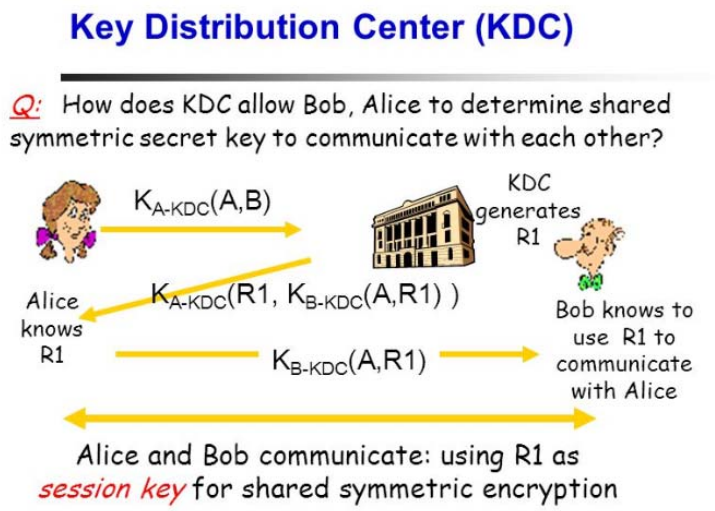
4x4 , 16 στοιχείων οπού καθένα είναι 1 byte – οπότε και ο πίνακας (τμήμα) αποτελείται από 128bit.



Εικόνα 2.3: Η σχηματική παρουσίαση των σταδίων του AES [30]

Στην κρυπτογραφία συμμετρικού κλειδιού, κρίσιμο στοιχείο αποτελεί η μυστικότητα του συμμετρικού κλειδιού. Αν αποκαλυφθεί σε τρίτους, αυτό το γεγονός καθιστά άκυρο τον όποιο αλγόριθμο συμμετρικού κλειδιού. Κατά συνέπεια, είναι ανάγκη να υπάρχουν αποτελεσματικοί τρόποι για την ασφαλή ανταλλαγή του κλειδιού μεταξύ των δύο μερών. Βεβαίως, έχουν γίνει κάποιες προσπάθειες αντιμετώπισης του θέματος αυτού. Η πρώτη, προφανής, προσέγγιση είναι με τη χρήση ενός ασφαλούς καναλιού επικοινωνίας και μετάδοσης του κλειδιού. Αυτό όμως θεωρείται εξ ορισμού μη ρεαλιστικό, αφού κατά κανόνα δεν υπάρχουν ασφαλή κανάλια μετάδοσης (αν υπήρχαν, δεν θα υπήρχε εξ αρχής η ανάγκη κρυπτογράφησης). Έχει

χρησιμοποιηθεί η αρχιτεκτονική των Κέντρων Διανομής Κλειδιών (Key Distribution Centers - KDC), τα οποία δεν είναι τίποτε παραπάνω από έναν εγγυητή της διαδικασίας. Με αυτή την μέθοδο, δυο συνομιλητές, για να ανταλλάξουν κλειδί, πιστοποιούνται εφάπαξ με ένα κύριο (master) κλειδί που διαμοιράζεται ο καθένας με το KDC. Μετά, για κάθε συνεδρία μεταξύ αποστολέα και παραλήπτη , ο καθένας εξ αυτών, στην ουσία απευθύνεται στο KDC και ζητά το κλειδί συνεδρίας (session key) για να συνομιλήσει με τον άλλο. Το KDC ελέγχει την εγκυρότητα και την ύπαρξη καθενός από τα δυο μέρη της συνοδού και κοινοποιεί το προσωρινό κλειδί συνεδρίας με ασφαλή τρόπο (αξιοποιώντας, για το κάθε μέλος, το κύριο κλειδί). Σε επόμενη συνεδρία το κλειδί αυτό είναι διαφορετικό, γεγονός σημαντικότατο καθώς πρόκειται για κλειδιά μιας χρήσεως. Παρακάτω βλέπουμε σχηματικά τη διαδικασία που προαναφέραμε



Εικόνα 2.4: Παράδειγμα εφαρμογής των KDC για την ανταλλαγή μυστικού κλειδιού συνεδρίας [27]

Στην πράξη, όλη αυτή η διαδικασία κρίνεται δυσλειτουργική αφού δε γίνεται σε ένα μεγάλο δίκτυο όπως το Internet, να βασίζονται όλες οι πιστοποιήσεις σε ένα κεντρικό KDC, αλλά σε σύστημα πολλών KDC με ιεραρχία και εμπιστοσύνη του ενός από το άλλο. Επίσης κάποιες φορές, όταν το session key λήγει , πρέπει να επαναλαμβάνεται ο αλγόριθμος μεταξύ των A, B και του KDC από την αρχή, εν μέσω συνεδρίας, καταναλώνοντας αρκετούς διαδικτυακούς πόρους αλλά και πόρους συστήματος.

Από την ανάγκη να αντιμετωπιστούν κάποια ζητήματα όπως αυτά της ανταλλαγής συμμετρικού κλειδιού, ανέκυψε μία άλλη κατηγορία κρυπτογραφικών αλγορίθμων, οι λεγόμενοι αλγόριθμοι δημοσίου κλειδιού.

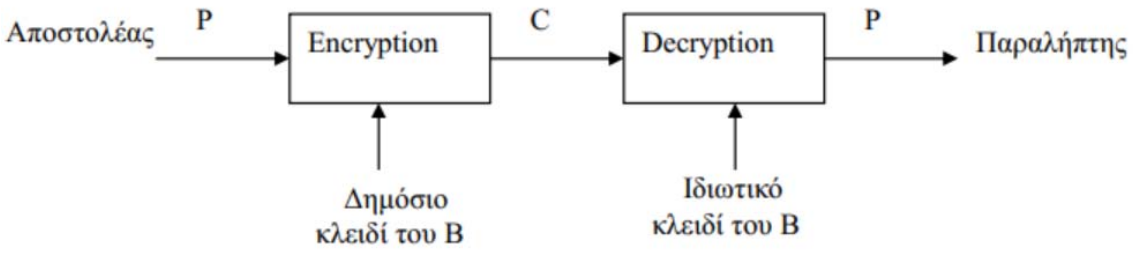
Κεφάλαιο 3
Συστήματα δημοσίου κλειδιού

3.1 Ιστορική αναδρομή και ευρέως διαδεδομένα συστήματα δημοσίου κλειδιού

Το 1976 οι επιστήμονες Diffie & Hellman πρότειναν μία νεωτεριστική μορφή κρυπτογράφησης που έμελλε να αλλάξει την συνολική εικόνα του μεταγενέστερου διαδικτύου[32]. Η γνωστή σε όλους κρυπτογράφηση δημόσιου κλειδιού περιλάμβανε μία νέα ιδέα για την ανταλλαγή συμμετρικού κλειδιού ακόμη και μέσα σε μη ασφαλές περιβάλλον (κανάλι επικοινωνίας). Σε αυτήν την περίπτωση το νέο αυτό κρυπτοσύστημα χρησιμοποιείται για την ανταλλαγή του συμμετρικού κλειδιού. Μετά την ανταλλαγή του συμμετρικού κλειδιού μπορεί να εφαρμόζεται η γνωστή κατά τα άλλα, μέθοδος συμμετρικής κρυπτογράφησης.

Η βασική φιλοσοφία των κρυπτοσυστημάτων δημοσίου κλειδιού είναι η εξής. Τόσο ο αποστολέας όσο και ο παραλήπτης έχουν παράξει ένα ζεύγος κλειδιών το οποίο αποτελείται από το ιδιωτικό και το δημόσιο κλειδί. Εν γένει, το συμμετρικό κλειδί κρυπτογραφείται από τον Α με το δημόσιο κλειδί του Β. Έπειτα το κρυπτογραφημένο αυτό μήνυμα, το οποίο περιέχει το μυστικό κλειδί, στέλνεται στον Β. Μόνο ο Β μπορεί να αποκρυπτογραφήσει αυτό το μήνυμα με την υπόθεση ότι το ιδιωτικό κλειδί του Β είναι γνωστό μόνο στον ίδιο. Για τον κάθε χρήστη, το ζεύγος δημόσιο – ιδιωτικό κλειδί είναι τέτοιο ώστε αν κρυπτογραφηθεί ένα μήνυμα με το δημόσιο κλειδί, τότε η αποκρυπτογράφησή του με το ιδιωτικό επιτρέπει την ανάκτηση του αρχικού μηνύματος – για αυτό και λέγονται και κρυπτοσυστήματα ασύμμετρου κλειδιού.

Όπως γίνεται κατανοητό, για την πραγματοποίηση αυτού του συστήματος, είναι απαραίτητο να υπάρχουν μαθηματικές συναρτήσεις κρυπτογράφησης και αποκρυπτογράφησης με αυτές τις ιδιότητες. Η αποκρυπτογράφηση επιστρέφει στην ουσία το καθαρό κείμενο και στην περίπτωση μας, το συμμετρικό κλειδί μόνο στον Β αφού κανένας άλλος δεν διαθέτει το ιδιωτικό κλειδί του Β. Παρόλα αυτά το Δημόσιο κλειδί του Α και του Β είναι γνωστό σε όλους. Τα κρυπτοσυστήματα αυτά κατασκευάζονται με τρόπο τέτοιο ώστε η γνώση του δημόσιου κλειδιού κάποιου χρήστη να μην επιτρέπει τον υπολογισμό του ιδιωτικού του κλειδιού, το οποίο δεν πρέπει να το γνωρίζει κανένας πλην του κατόχου του.



Εικόνα 3.1: Γενική σχηματοποίηση Public Key Crypto System [25].

Οι αλγόριθμοι δημόσιου κλειδιού βασίζουν την ασφάλειά τους σε γνωστά «δύσκολα» μαθηματικά προβλήματα. Η «δυσκολία» επίλυσης των προβλημάτων αυτών έγκειται σε γνωστούς ορισμούς της θεωρίας πολυπλοκότητας [23].

3.2 Το ζήτημα της παραγοντοποίησης ενός (μεγάλου) αριθμού

Αυτό το μαθηματικό πρόβλημα κατέχει μία εξαιρετικά σημαντική θέση στον τομέα της σύγχρονης κρυπτογραφίας. Αρκετά πρωτόκολλα, κυρίως δημόσιου κλειδιού, βασίζονται στην πολυπλοκότητα της επίλυσης του συγκεκριμένου ζητήματος. Καθώς οι απαιτήσεις για την ασφάλεια του διαδικτύου, σήμερα περισσότερο από ποτέ, λαμβάνονται υπόψη από τους επιστήμονες, η εύρεση των πρώτων παραγόντων ενός μεγάλου αριθμού και το σχετικό πρόβλημα καθορίζουν τους εγγυητές ασφαλείας των πρωτοκόλλων που βασίζονται σε αυτό το μαθηματικό θέμα. Πιο συγκεκριμένα το πρόβλημα παραγοντοποίησης ενός μεγάλου αριθμού είναι το εξής :

Έστω αριθμός που δεν είναι πρώτος. Να βρεθεί ένας πρώτος παράγοντάς του., δηλαδή ένα πρώτος αριθμός που τον διαιρεί ακριβώς. Είναι γνωστό ότι κάθε αριθμός γράφεται κατά μοναδικό τρόπο ως γινόμενο πρώτων αριθμών (οι οποίοι λέγονται πρώτοι παράγοντες (prime factors) αυτού), και το πρόβλημα παραγοντοποίησης έγκειται στο να βρεθούν οι πρώτοι παράγοντες ενός αριθμού.

Για την λύση του δοθέντος προβλήματος, μία πρώτη αντιμετώπιση είναι η μέθοδος δοκιμής διαιρετών, η οποία γίνεται με διαδοχικές δοκιμές αριθμών από το δύο έως την τετραγωνική ρίζα του αριθμού αυτού μέχρι να βρεθεί κάποιος διαιρέτης του. Αυτό συμβαίνει βάσει της παραδοχής ότι κάθε μη πρώτος αριθμός πρέπει να έχει ένα διαιρέτη μικρότερο της τετραγωνικής ρίζας του. Με τη μέθοδο αυτή έχουμε πολυπλοκότητα O (√n) και αφού σε προβλήματα θεωρίας αριθμών, το μέτρο της χρονικής πολυπλοκότητας είναι τα bits του n , αυτή ισοδυναμεί με $O(\log(2^n))$ και άρα είναι εκθετικού χρόνου. Φυσικά υπάρχουν κι άλλες μέθοδοι, επίσης όμως με πολυπλοκότητα εκθετικού χρόνου .

Γενικότερα, με τα σημερινά υπολογιστικά συστήματα, το πρόβλημα της παραγοντοποίησης ενός μεγάλου αριθμού (π.χ. της τάξης των 2048 bits) θεωρείται δύσκολο να επιλυθεί.

3.3 Πρόβλημα Διακριτού λογαρίθμου

Θεωρούμε δεδομένο ότι είναι εύκολο να υπολογιστεί η δύναμη b^X σε σχετικά μικρό χρόνο.

Έστω κάποιος αριθμός που έχει την μορφή y=b^X και το b είναι γνωστό σε εμάς.

Τότε για να βρούμε τον αριθμό χ πρέπει να υπολογίσουμε τον λογάριθμο $\log_b(y)$. Η λύση αυτής της εξίσωσης όταν οι πράξεις γίνονται modulo p για κάποιον πρώτο αριθμό p είναι και το πρόβλημα Διακριτού λογάριθμου. Η εύρεση της λύσης του προβλήματος είναι αρκετά δύσκολη αν επιλέξουμε κατάλληλα το πεπερασμένο σώμα (κυκλική ομάδα) στο οποίο αναφερόμαστε, ήτοι τον πρώτο αριθμό p .

Πιο συγκεκριμένα, το πρόβλημα διακριτού λογαρίθμου σε πεπερασμένα σώματα περιγράφεται στη συνέχεια.

Κατ' αρχάς, ένας αριθμός g ονομάζεται γεννήτορας mod p αν όλες οι δυνάμεις

$g^1(mod\ p)\ g^2(mod\ p)\ \ldots\ldots g^{p-1}(mod\ p)$

είναι ανά δύο διαφορετικές μεταξύ τους.

Zp^* ή αλλιώς GF(p) ορίζουμε ένα πεπερασμένο σώμα, όπου p πρώτος αριθμός.

Ειδικότερα, για το Πρόβλημα Διακριτού Λογαρίθμου (DLOG) ο ορισμός είναι ο εξής:

Δίνονται: ένας πρώτος αριθμός p, ένας γεννήτορας α του Zp^* και ένα στοιχείο $\beta \in Zp^*$.

Ζητείται: Να βρεθεί ακέραιος x, $0 \le x \le p - 2$, τέτοιος ώστε $\alpha^X = \beta \pmod p$

3.4 Αλγόριθμος Diffie-Hellman

Ο αλγόριθμος αυτός εκμεταλλεύεται την δυσκολία του προβλήματος διακριτού λογαρίθμου και παρουσιάζεται στη συνέχεια.

Η φιλοσοφία του αλγορίθμου αυτού είναι η ασφαλής μεταφορά ενός και μόνο αριθμού μεταξύ δύο χρηστών Α και Β. Το πλεονέκτημα είναι ότι δεν απαιτείται γνώση κάποιας άλλης πληροφορίας. Ο αλγόριθμος αυτός βασίζεται στην δυσκολία του προβλήματος διακριτού λογαρίθμου και δεν μπορεί να κρυπτογραφήσει ένα μήνυμα αλλά μόνο να παράξει ένα συμμετρικό κλειδί που θα ανταλλαγει με ασφάλεια σε 2 μέρη.

Ο αλγόριθμος Diffie-Hellman λειτουργεί ως εξής:

i. Οι δύο χρήστες Α και Β συμφωνούν σε έναν πρώτο αριθμό (κατά σύμβαση αρκούντως μεγάλο) p και έναν αριθμό g που είναι γεννήτορας (mod p).

ii. Η δυάδα p, g αποτελεί το Δημόσιο κλειδί του αλγορίθμου.

iii. Α επιλέγει έναν μυστικό αριθμό x.

iv. Β επιλέγει έναν μυστικό αριθμό y.

v. Ο Α στέλνει στον Β τον αριθμό x'=g^x (mod p)

vi. Ο Β στέλνει στον Α τον αριθμό y' =g^y (mod p)

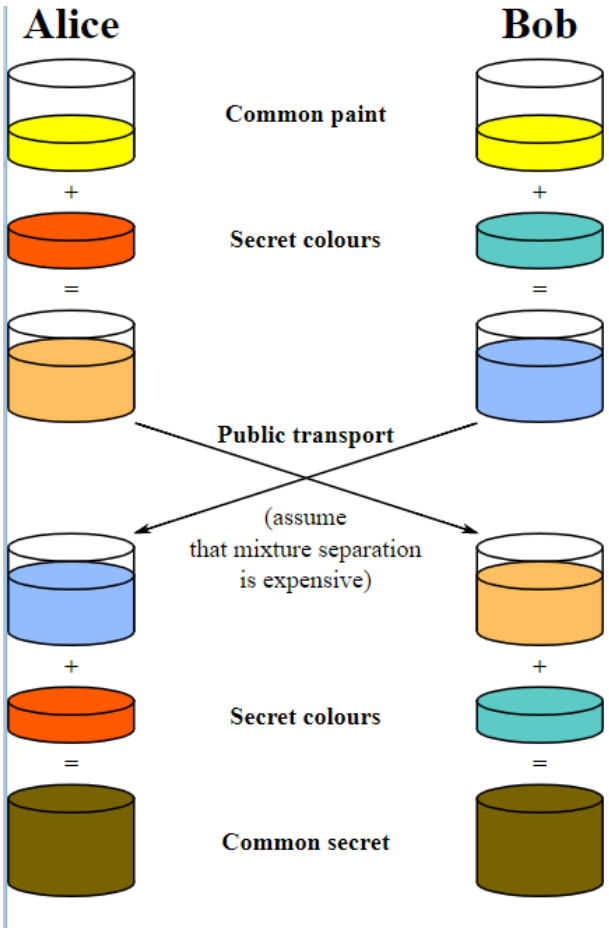
- Α υπολογίζει τον y'^x (mod p)= g^{yx} (mod p)

- Β υπολογίζει τον x'^y (mod p)= g^{yx} (mod p)

vii. Και οι δύο έχουν με απόλυτα μυστικό τρόπο αποκτήσει τον ίδιο αριθμό g^{yx} (mod p), ο οποίος μπορεί να χρησιμοποιηθεί και ως συμμετρικό κλειδί για τη συνέχεια της επικοινωνίας.

Συμπερασματικά, με δημόσια τα p, g παράγεται με επιτυχία και στις δυο πλευρές το μυστικό (ιδιωτικό) κλειδί g^{yx} (mod p). Κανείς επίδοξος υποκλοπέας, που παρακολουθεί την ανταλλαγή μηνυμάτων, δεν μπορεί να υπολογίσει το μυστικό αυτό κλειδί: θα μπορούσε μόνο αν γινόταν να επιλύσει το πρόβλημα διακριτού λογαρίθμου.

Ο αλγόριθμος αυτός χρησιμοποιείται πλέον ευρέως για τα πρωτόκολλα IPSec, TLS,/https. Φυσικά ο Α και ο Β πρέπει να πιστοποιούν τον εαυτό τους, υπό την έννοια ότι ο Α πρέπει να είναι σίγουρος για την ταυτότητα του Β και αντίστροφα. Η κρυπτογραφία παρέχει μηχανισμούς και για αυτό το ζήτημα, όπως οι ψηφιακές υπογραφές και τα ψηφιακά πιστοποιητικά [16]. Παρακάτω παραθέτουμε την απεικόνιση του αλγορίθμου Diffie-Hellman ως πρωτότυπο σχέδιο για την καλύτερη κατανόηση του αλγορίθμου.



Εικόνα 3.2: Σχηματοποίηση με χρώματα του αλγορίθμου Diffie-Hellman [30]

3.5 Κρυπτοσύστημα δημοσίου κλειδιού RSA

Ένας πολύ γνωστός κρυπτογραφικός αλγόριθμος δημοσίου κλειδιού είναι ο RSA, ο οποίος πηρέ το όνομα του από τους Rivest, Shamir,Adleman, οι οποίοι και τον προτείναν [33]. Για την κατανόηση του αλγοριθμου αυτού, μελετούμε συνοπτικά κάποιες σχετικές μαθηματικές έννοιες. Συνάρτηση Euler φ(N) ακεραίου αριθμού N είναι ίση με το πλήθος των θετικών αριθμών που είναι μικρότεροι του N και είναι πρώτοι ως προς το N , συνεπώς δεν έχουν κοινούς διαιρέτες με τον αριθμό N.

Αν θέσουμε ως παράδειγμα τον αριθμό N=12 τότε οι

gcd(1,12) = gcd(5,12) = gcd(7,12) = gcd(11,12)=1 άρα, φ(12)=4

Σύμφωνα με το λεγόμενο Θεώρημα Euler, αν gcd(m, N)=1, τότε $m^{\varphi(N)} \equiv 1 \pmod N$

Οι ανωτέρω ιδιότητες έχουν ιδιαίτερη σημασία για το κρυπτοσύστημα RSA.

Συγκεκριμένα, ο αλγόριθμος έχει ως εξής:

i. Ένας χρήστης Α επιλέγει με τυχαίο τρόπο δύο μεγάλους πρώτους αριθμούς p και q.

ii. Ο Α υπολογίζει το γινόμενο N = p q το όποιο θα ήταν καλό να αποτελείται από τουλάχιστον 2048 ή, ακόμα καλύτερα, 3072 ψηφία για λόγους ασφάλειάς.

iii. Ο Α προχωρά στον Υπολογισμό του φ(N) , και αφού οι p, q πρώτοι αποδεικνύεται ότι ισχύει:

φ(N)=(p-1)(q-1)

iv. Ο Α επιλέγει τυχαίο αριθμό e, τέτοιο ώστε gcd(e, φ(N))=1

v. Ο Α υπολογίζει το d=e^{-1} (mod φ(N)) που αποτελεί και το ιδιωτικό του κλειδί. Με άλλα λόγια, ο d είναι ο μοναδικός ακέραιος αριθμός με την ιδιότητα $e^d \equiv 1 \pmod{(\varphi(N))}$.

vi. Ο Α αποστέλλει προς κάθε ενδιαφερόμενο το Δημόσιο κλειδί του που είναι το ζεύγος (N, e)

vii. Εφόσον το d έχει προέλθει από υπολογισμούς στα αρχικά δεδομένα p, q, φ(N), για αυτά κρίνεται ως απαραβίαστη σύμβαση να μείνουν μυστικά.

viii. Ο Β έχει λάβει το δημόσιο κλειδί (N, e) του Α.

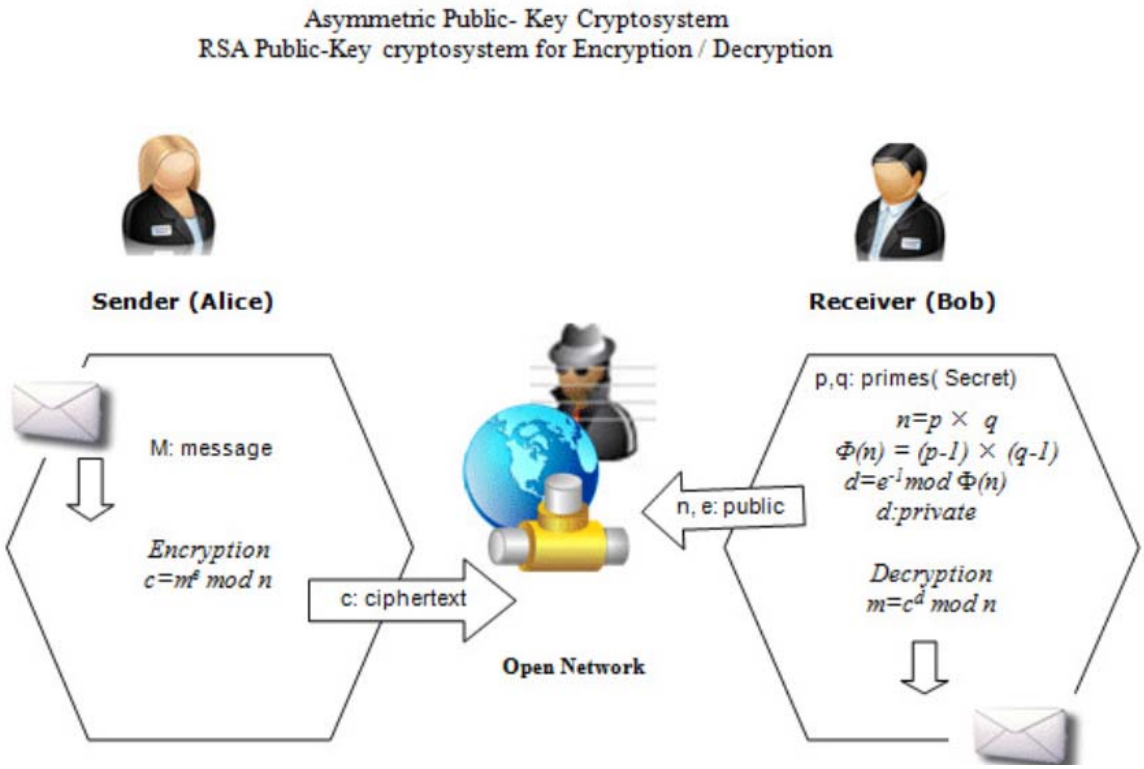
ix. Ο Β για να κρυπτογραφήσει ένα μήνυμα M σε ένα κρυπτοκείμενο C το οποίο να μπορεί να το αποκρυπτογραφήσει ο Α εκτελεί την εξής πράξη:

C = $M^e \pmod N$

x. Ο Β μπορεί να αποστείλει το μήνυμα σε μη ασφαλές κανάλι επικοινωνίας, αφού μόνο ο Α με το ιδιωτικό κλειδί του μπορεί να το αποκρυπτογραφήσει.

xi. Ο Α για την αποκρυπτογραφήσει του C, εκτελεί την εξής πράξη:

M= C^{d} (mod N)



Εικόνα 3.3: Αναπαράσταση του σχήματος κρυπτογράφησης RSA [30]

3.6 Ασφάλεια του RSA

3.6.1 Επίθεση κρυπτανάλυσης απευθείας στο δημόσιο κλειδί N.

Με την παραγοντοποίηση του N, ο εισβολέας , μπορεί να βρει τα p, q και από αυτά καταλήγει άμεσα στο φ(N)=(p-1)(q-1), άρα και στο ιδιωτικό κλειδί του Α από τη σχέση d=e^{-1} (mod φ(N)) όπως αυτή προαναφέρθηκε.

Ωστόσο, η παραγοντοποίηση του N, στην οποία βασίζεται η ασφάλεια του RSA, είναι ένα δύσκολο μαθηματικό πρόβλημα για μεγάλες τιμές του N. Από αυτό γίνεται ευθέως κατανοητή η ανάγκη για χρήση αριθμών μεγάλου μήκους, καθώς οι αλγόριθμοι παραγοντοποίησης συνεχώς βελτιώνονται ανάλογα και με την ισχύ των υπολογιστικών μηχανών αλλά και την εξέλιξή της τεχνολογίας. Το ελάχιστο ασφαλές μέγεθος κλειδιού κρίνεται ότι είναι προς το παρόν τα 2048 bit. Έχει αναφερθεί παραγοντοποίηση πρώτου αριθμού με περισσότερα από 600 bit [09].

3.6.2 Επιθέσεις χρονισμού

Μια κατηγορία επιθέσεων στον αλγόριθμο RSA – η οποία βέβαια υφίσταται και σε άλλους κρυπτογραφικούς αλγορίθμους - είναι οι επιθέσεις παράπλευρου καναλιού (Side Channel Attack), στις οποίες ο εισβολέας προσπαθεί να εξάγει χρήσιμα συμπεράσματα για το ιδιωτικό κλειδί , κάνοντας μετρήσεις φυσικών ποσοτήτων, όπως η ταχύτητα και η ενεργειακή κατανάλωση από κάποιο αλγόριθμο στο τοπικό μηχάνημα. Τέτοιου τύπου είναι και οι λεγόμενες επιθέσεις χρονισμού (timing attacks), που ορίζονται ως οι επιθέσεις που βασίζονται στην παρατήρηση της χρονικής καθυστέρησης της αποκρυπτογράφησης ενός RSA κρυπτοκειμένου από κάποιον χρήστη Α. Γνωρίζοντας τον τύπο και τα χαρακτηριστικά του λογισμικού και υλισμικού του Α, θα μπορούσε κάποιος να συμπεράνει το μέγεθος του d. Έτσι θα έχει περιορίσει σημαντικά το εύρος αριθμών προς εξερεύνηση για τα p, q και θα μπορούσε ενδεχομένως να κατακτήσει το ιδιωτικό κλειδί. Αυτό αντιμετωπίζεται με εισαγωγή τυχαίας ψευδό καθυστέρησης σε επίπεδο αλγορίθμου. Έτσι ο χρόνος αποκρυπτογράφησης είναι τυχαίος και δεν μπορεί κάποιος να εξάγει ασφαλή συμπεράσματα για τα μεγέθη p, q. Αυτό δε σημαίνει πως η τεχνική χρησιμοποιείται μόνο για τον RSA αλλά επεκτείνεται και σε κρυπτογραφία συμμετρικού κλειδιού [33].

3.6.3 Επιθέσεις επιλεγμένου κρυπτοκειμένου (chosen ciphertext attack - CCA)

Στις επιθέσεις αυτής της κατηγορίας, γίνεται η παραδοχή ότι ο εισβολέας έχει τη δυνατότητα επιλογής κρυπτοκειμένου C για το όποιο μπορεί να μάθει πώς θα αποκρυπτογραφούνταν με τη χρήση του μυστικού κλειδιού, το όποιο σκοπεύει να βρει.

Έστω ένα τέτοιο κρυπτοκείμενο c = m^e (mod N)

Δεχτήκαμε de facto (CCA Attack) ότι ο εισβολέας έχει τη δυνατότητα να ξέρει κάθε φορά ποιο είναι το μήνυμα m' που αν κρυπτογραφηθεί με βάση το δημόσιο κλειδί (e, N) θα έχει ως αποτέλεσμα το c. Κατά τα γνωστά ισχύει ότι c= m'^{e} (mod N) και m' = c^{d} (mod N) χωρίς να γνωρίζει όμως το d.

Ο επιτιθέμενος, γνωρίζοντας το c, βρίσκει το x =c 2^{e} (mod N) =c $(mod\ N)$ (2^{e} (mod N))= m^{e} (mod N) (2^{e} (mod N))= $[2m]^{e}$ (mod N) . Είναι σε θέση, από την παραδοχή μας, να μάθει πώς θα αποκρυπτογραφούνταν το x – άρα, είναι σε θέση να μάθει το 2m (αφού η αποκρυπτογράφηση του x για το συγκεκριμένο άγνωστο κλειδί d θα ισούται με 2m, όπως μόλις αποδείξαμε). Ως εκ τούτου, τελικά με γνωστό το c ο εισβολέας βρίσκει το μήνυμα m κάνοντας επίθεση CCA.

Εδώ αξίζει να αναφέρουμε ότι υπάρχουν διάφορα κριτήρια αποτίμησης της ασφάλειας, έτσι ώστε ακόμα και με παραδοχές όπως ανωτέρω να αποδεικνύεται ότι ο επιτιθέμενος δεν μπορεί τελικά να καταλήξει σε επιτυχή κρυπτανάλυση. Για παράδειγμα, ένα κρυπτοσύστημα παρέχει σημασιολογική ασφάλεια (semantic security), εφόσον ισχύει το εξής: ακόμα και αν ο υποκλοπέας, για δοθέν κρυπτοκείμενο, γνωρίζει ότι προέρχεται από ένα μεταξύ δύο γνωστών μηνυμάτων, δεν μπορεί να αποφανθεί ποιο από τα δύο μηνύματα είναι τελικά αυτό που κρυπτογραφήθηκε. Οι αλγόριθμοι δημοσίου κλειδιού, λόγω του ότι χρησιμοποιούν ένα δημόσιο κλειδί για την κρυπτογράφηση, δεν μπορούν να είναι σημασιολογικά ασφαλείας παρά μόνο αν, κάθε φορά που γίνεται κρυπτογράφηση ενός μηνύματος, υπεισέρχεται και μία τυχαία ποσότητα, έτσι ώστε ακόμα και αν κρυπτογραφηθεί το ίδιο μήνυμα δύο φορές θα προκύπτει κάθε φορά διαφορετικό κρυπτοκείμενο (βέβαια, αυτό είναι αναγκαία μεν αλλά όχι ικανή συνθήκη). Η διαδικασία αυτή εισήχθη στο πρότυπο PKCS (Public Key Cryptography Standards). Η RSA security inc. προτείνει το Optimal Asymmetric Encryption Padding (OAEP) που υλοποιείται στο PKCS1v2.2 για την ικανοποίηση της ως άνω ιδιότητας [33]. Η υλοποίηση του RSA βάσει του προτύπου αυτού δεν είναι ευάλωτη σε επιθέσεις CCA.

3.6.4 Επίθεση εκμεταλλευομένη βασικές παραμέτρους.

Πρέπει, όπως προαναφέρθηκε, οι αριθμοί p, q να είναι αρκετά μεγάλοι αλλά και το |p - q| να μην είναι πολύ μικρό. Αν το τελευταίο δεν υλοποιείται σωστά, τότε ο εισβολέας θα μπορούσε να πέτυχει τις τιμές p, q κάνοντας δοκιμές στην αριθμητική περιοχή κοντά στο √N .

Τα e, d πρέπει να μην είναι μικρού μεγέθους. Όταν το d είναι μικρότερο από $N^{1/4}$ τότε υπάρχει τεχνική υπολογισμού του d [09].

Έχει υπάρξει ακόμη και περιστατικό εύρεσης των πρώτων αριθμών p, q με δεδομένα τα N, e , d, οπότε πρέπει να αποφεύγεται η χρήση ιδίου N για μεγάλες ομάδες χρηστών κατά τη δημιουργία ζεύγους κλειδιού RSA [09].

Κεφάλαιο 4
Κβαντικοί υπολογιστές

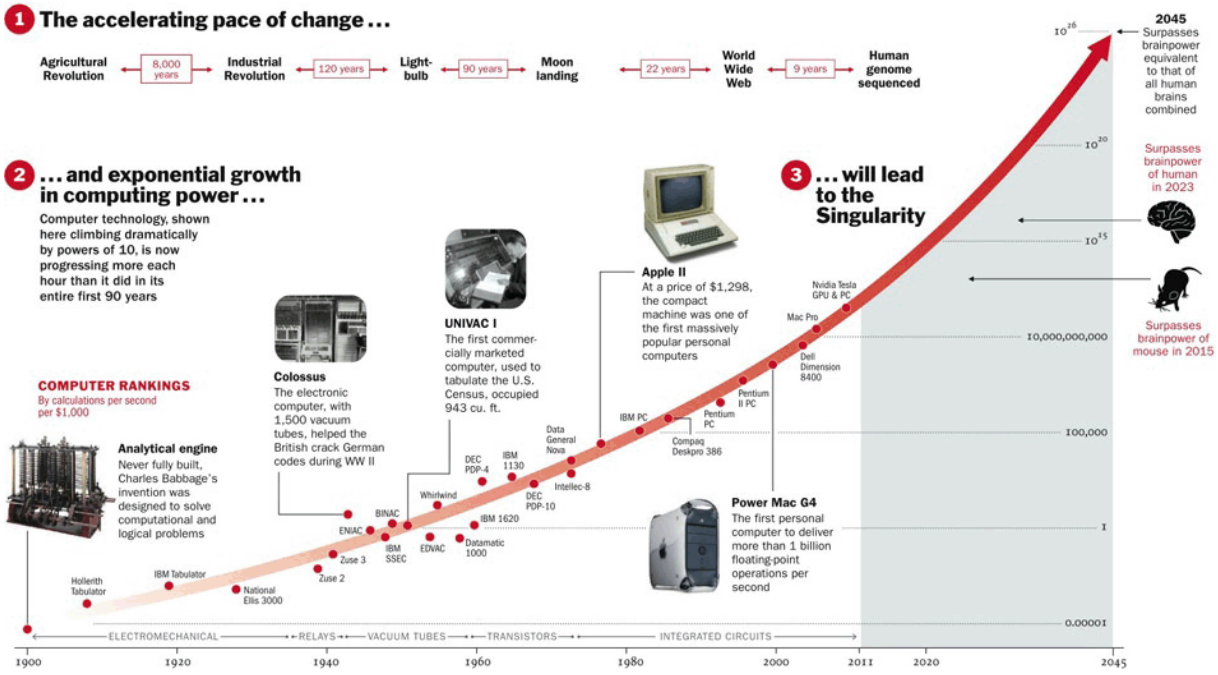
4.1 Ιστορικά Στοιχεία

Τις τελευταίες δεκαετίες γίνεται συχνά λόγος για την υλοποίηση ενός συστήματος κβαντικού υπολογιστή. Αυτό δύναται να αλλάξει πολλά στο τοπίο της πληροφορικής και των μαθηματικών, αλλά και της κρυπτογραφίας.

Αρχικά, ως κβαντικός υπολογιστής ορίζεται οποιαδήποτε συσκευή η οποία χρησιμοποιεί φαινόμενα της κβαντομηχανικής ώστε να κάνει υπολογισμούς και να διαχειριστεί δεδομένα[13]. Σε αντίθεση με τον κλασσικό υπολογιστή ή αλλιώς υπερυπολογιστή, τα κυκλώματα του κβαντικού υπολογιστή συνυπάρχουν με τα κβαντικά αυτά φαινόμενα, τα οποία είναι στοχαστικά

και όχι ντετερμινιστικά. Δηλαδή οι πύλες δεν υλοποιούνται με απλά τρανζίστορ αλλά βασίζονται σε κβαντομηχανικές συναρτήσεις κύματος.

Το θεώρημα Moore υποστηρίζει ότι κάθε 2 χρόνια ο αριθμός των τρανζίστορ σε ένα chip πυριτίου διπλασιάζεται. Ήδη οι εταιρείες όπως η Intel έχουν φτάσει σε στάσιμη κατάσταση (plateau) συνεπώς ο νομός Moore φαίνεται να έχει επισκιαστεί από τις χαμηλότερες επενδύσεις στον τομέα αυτό[24]. Αυτό δεν μπορεί να συμβαίνει επ' άπειρον. Υπάρχει ένα σαφές όριο– αυτό του μεγέθους του ατόμου. Εν ολίγοις, ένα τρανζίστορ κάποια στιγμή αναμένεται να φτάσει στο μέγεθος ενός ατόμου, και αυτό θα σημάνει ίσως το τέλος μιας εποχής εξέλιξης των διαστάσεων των πυλών προς το μικρότερο φυσικά. Μετά από αυτό το σημείο εισαγόμαστε σε κβαντικά φαινόμενα - άρα και στον κβαντικό υπολογιστή; Θα μπορούσε κανείς να το υποστηρίξει αυτό. Πιο κάτω βλέπουμε το νομό Moore σε δεδομένα απεικονισμένα κατάλληλα.



Εικόνα 4.1: Απεικόνιση του νόμου Moore σε ορίζοντα 30 ετών [24]

Πλησιάζοντας αρκετά το μέγεθος του ατόμου, προτείνονται διάφορες εναλλακτικές καθώς βαίνουμε το κατώφλι των κβαντικών φαινομένων. Φυσικά υπάρχουν ακόμη πολλές προσεγγίσεις τεχνολογικά, όπως τρανζίστορ στα 1,5 nm, αλλά και τρανζίστορ που αποτελείται από ένα μοναδικό άτομο τοποθετημένο επάνω σε βάση πυριτίου. Η IBM το 2015 έκανε επίδειξη ενός

τρανζίστορ 7nm γερμανίου & πυριτίου. Ενδεχομένως κάποιες πρωτοποριακές τεχνολογίες να βοηθήσουν στην χρονική επιμήκυνση της ισχύος του νόμου Moore. Παρόλα αυτά μετά από κάποιες δεκαετίες η αρχιτεκτονική των τρανζίστορ θα προσκρούσει και πάλι στις ατομικές διαστάσεις και στα κβαντικά φαινόμενα γύρω από αυτές. Ίσως ο κβαντικός υπολογιστής είναι μια διέξοδος και με αυτό τον τρόπο επωφελούμαστε από τις ιδιότητες των κβαντικών φαινομένων αντί να μας δυσκολεύουν στην πρόοδο των υπολογιστών. Το δίκτυο IBMQ (https://www.research.ibm.com/ibm-q/) ενώνει την τεχνογνωσία πολλών εταιρειών και πανεπιστημιακών ιδρυμάτων ανά τον κόσμο για να φέρει εις πέρας την αποστολή της κατάκτησης του κβαντικού υπολογιστή.

Στον μέχρι σήμερα κλασσικό υπολογιστή, όλοι οι αλγόριθμοι που είναι βασισμένοι σε οποιαδήποτε γλώσσα προγραμματισμού, τελικώς, κατά την επεξεργασία τους και την εκτέλεσή τους καταλήγουν σε δυαδικού τύπου εντολές και τη διαχείριση των δεδομένων σε δυαδικά ψηφία (bits). Η δυαδική λογική είναι μέχρι τώρα διαδεδομένη στην επιστήμη των υπολογιστών. Εδώ οι υπολογισμοί γίνονται με τη βοήθεια ενός αλγορίθμου που είναι ένα σύνολο από οδηγίες προς επίλυση συγκεκριμένου προβλήματος.

4.2 Διαφορές υπολογιστή και φυσικού φαινομένου

Ένας υπολογιστής απαντά σε μαθηματικές ερωτήσεις και ζητήματα. Ένα πείραμα φυσικής απαντά σε ερωτήσεις που αφορούν φυσικά φαινόμενα. Αυτό μας δίδει την διαφορά μεταξύ ενός πειράματος φυσικής και ενός υπέρ-υπολογιστή.

Πέρα από αυτό, ένα φυσικό πείραμα είναι κατά κανόνα μία μεγάλης κλίμακας κατασκευή. Ο υπολογιστής είναι ένα απλό κουτί μικρών διαστάσεων που βρίσκεται σε ένα συγκεκριμένο χώρο.

Άλλη μία ειδοποιός διαφορά ανάμεσα στον υπολογιστή και το φυσικό φαινόμενο είναι ότι δεν χρειάζεται κάθε φορά που θέλουμε να λύσουμε ένα διαφορετικό μαθηματικό πρόβλημα να κατασκευάζουμε ένα διαφορετικό ηλεκτρονικό υπολογιστή.

4.3 Βασικοί ορισμοί

Μία μηχανή Turing μπορεί να κάνει οποιοδήποτε υπολογισμό υπερκαλύπτοντας κάθε άλλη φυσική μηχανή. Δηλαδή μπορεί να κάνει οποιοδήποτε υπολογισμό οποιασδήποτε άλλης μηχανής.

Για λόγους απλότητας θεωρούμε ότι ένα πρόγραμμα πρέπει να υπολογίζει μία συνάρτηση-αλγόριθμο σε ένα λογικό χρονικό διάστημα - το πολύ ετών. Αλλιώς η συνάρτηση ή η διαδικασία δεν θεωρείται υπολογίσιμη. Οι θεωρητικοί επιστήμονες πιστεύουν ότι ένας αλγόριθμος είναι πρακτικός εάν ο χρόνος στον οποίο εκτελείται είναι μία πολυωνυμική συνάρτηση του μεγέθους N της εισόδου του προγράμματος.

Το σύνολο των προβλημάτων που λύνονται σε πολυωνυμικό χρόνο λέγεται P και δεν εξαρτάται από το είδος υπολογιστικής μηχανής που χρησιμοποιείται για την λύση τους.

Αυτό οδήγησε στην θέση του Church ο όποιος αναθεωρώντας το universality of computation δήλωσε ότι μία μηχανή Turing μπορεί να εκτελεί αποδοτικά οποιοδήποτε υπολογισμό τον οποίον μπορεί να εκτελέσει οποιαδήποτε άλλη φυσική συσκευή το ίδιο αποδοτικά.

Στην πραγματικότητα η θέση του Church εμποδίζει κάποιες φορές την σωστή κατανόηση της χρήσης και της λειτουργικότητας των κβαντικών υπολογιστών. Μάλιστα ένα απλό βήμα εντολής σε ένα κβαντικό υπολογιστή μπορεί να είναι περισσότερο αργό από την εκτέλεση του ίδιου βήματος σε συμβατικό υπολογιστή. Στην πραγματικότητα οι κβαντικοί υπολογιστές επιταχύνουν τους υπολογισμούς με δραστική μείωση του αριθμού των βημάτων που χρειάζονται για να εκτελεστούν οι υπολογισμοί.

Η εξομοίωση φυσικών φαινομένων από έναν συμβατικό υπολογιστή φαντάζει αρκετά μη αποδοτική, σε αντίθεση με το κβαντικό υπολογιστή που μπορεί να υπερπηδήσει αυτό το εμπόδιο.

Στην παρούσα εργασία μας ενδιαφέρει κατά πόσον οι κβαντικοί υπολογιστές μπορούν να επιταχύνουν τον υπολογισμό ζητημάτων που δεν είναι σχετικά με την Κβαντική μηχανική αλλά με την κρυπτογραφία.

Συγκεκριμένα ξέρουμε ότι οι κβαντικοί υπολογιστές θεωρητικά είναι καλοί στο να:

i. υλοποιούν την εξομοίωση μηχανικών συστημάτων που άπτονται της κβαντικής μηχανικής.

ii. μπορούν να λύνουν πιο εύκολα προβλήματα παραγοντοποίησης μεγάλων αριθμών ή και το πρόβλημα του διακριτού λογαρίθμου.

iii. βρίσκουν την περιοδικότητα σε σχετικά προβλήματα.

iv. αναζητούν σε χώρους διανυσμάτων μεγάλων διαστάσεων (αλγόριθμος Grover [11]).

Ως προς το iv, οι κβαντικοί υπολογιστές δίνουν μία μεγάλη επιτάχυνση σε σχέση με προβλήματα εξαντλητικής αναζήτησης. Για να εξετάσει κάνεις το πρόβλημα εύρεσης ενός σημείου από N πιθανά σημεία χρειάζεται N/2 χρόνου σε ένα κλασικό υπολογιστή και π/4 √N σε ένα κβαντικό υπολογιστή. Επιπλέον ως προς το σημείο ii, ο κβαντικός υπολογιστής δίνει μία εκθετική επιτάχυνση στα προβλήματα παραγοντοποίησης μεγάλων αριθμών.

4.4 Μηχανή Turing

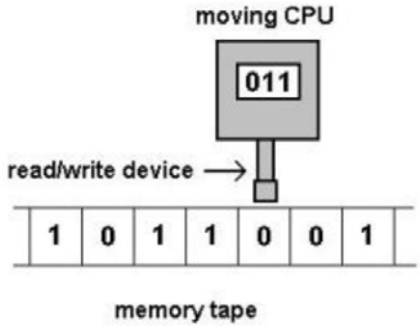
Ο Alan Turing δημιούργησε μια υποθετική μηχανή που θα μπορούσε να εκτελεί κάθε αλγόριθμο. Το όνομά της είναι Μηχανή Turing ή αλλιώς TM (Turing Machine) [13]. Αποτελείται από:

i. μια ταινία η οποία περιέχει άπειρο αριθμό θέσεων, που ξεκινούν από αριστερά και εκτείνονται προς το άπειρο.

ii. Μια κεφαλή που έχει έναν πεπερασμένο αριθμό καταστάσεων και την κατάσταση «halt»

iii. Η κεφαλή ανάγνωσης-εγγραφής που μπορεί να γραφεί να αναγιγνώσκει αλλά και να διαγράφει ένα ψηφίο κάθε φορά, αλλά και να μετακινείται αριστερά και δεξιά πάνω στην ταινία.

Μία σχηματική αναπαράσταση της μηχανής Turing φαίνεται παρακάτω:



Εικόνα 4.2: Σχηματική αναπαράσταση της μηχανής Turing [13]

Κάθε πρόγραμμα για μια ΤΜ γράφεται ως σύνολο εντολών. Μια εντολή γράφεται ως εξής :

T:(s, a) → (s', a', d), όπου:

i. s: η παρούσα κατάσταση

ii. s': τελική θέση κεφαλής

iii. a: το γράμμα στην παρούσα θέση

iv. a': το γράμμα που θα εγγραφεί στην τελική θέση

v. d: η φορά στην οποία θα μετακινηθεί η κεφαλή

Είσοδοι (Input) της μηχανής Turing: α) Ένας συγκεκριμένος αριθμός μη κενών θέσεων στην ταινία, β) η κεφαλή εγγραφής και ανάγνωσης σε συγκεκριμένη αρχική θέση, γ) η κεφαλή ελέγχου σε μια αρχική κατάσταση, δ) ένα σύνολο εντολών για κάθε επόμενη κατάσταση.

Έξοδος (Output) της μηχανής Turing: η τελική κατάσταση - αν αυτή υπάρχει. Η κεφαλή ελέγχου φτάνει το σημείο halt ενώ υπάρχουν ακόμη κενές θέσεις στην ταινία.

Σε μια μηχανή Turing είναι αδύνατο να γνωρίζουμε για όλα τα δυνατά input αν η κεφαλή ελέγχου θα φτάσει σε κατάσταση halt - γνωστό ως «halting problem».

Αν υποθέσουμε ότι η αρχική κατάσταση και οι παράμετροι εισόδου είναι στον πίνακα παρακάτω:

s	a	s'	a'	d
S1	b	S2	b	L
S2	b	S3	b	L

S2	1	S2	1	L
S3	b	H	b	0
S3	1	S4	b	R
S4	b	S2	1	L

Πίνακας 4.1: Στον πίνακα φαίνεται η αρχική κατάσταση και οι εντολές της μηχανής Turing.

Η είσοδος είναι γνωστή και είναι η :

b	b	1	1	b	1	1	1	b	b

Πίνακας 4.2: Ηείσοδος της μηχανής Turing

Η έξοδος θα είναι:

b	b	b	1	1	1	1	1	b	b

Πίνακας 4.3: Η έξοδος της μηχανής Turing

Για να γίνουν όλοι οι υπολογισμοί αυτόματα χρειάζονται κυκλώματα τα οποία αποτελούνται από καλώδια και πύλες. Κάθε γραμμή μεταφέρει ένα bit πληροφορίας. Οι πύλες επιτελούν λογικές πράξεις αναμεσά σε δύο η περισσότερες εισόδους κατά την άλγεβρα Bool. Σε κάθε υπολογισμό η είσοδος σε μια πύλη μπορεί να αναπαρασταθεί ως n bits, ενώ η έξοδος ως m bits:

$$C: \{0,1\}^n \rightarrow \{0,1\}^m$$

Οι πύλες AND, OR, NOT, FANOUT αποτελούν ένα καθολικό σύνολο από προκαθορισμένες λογικές πύλες που σημαίνει ότι κάθε λογική πράξη τύπου n-bit προς m-bit μπορεί να αναπαρασταθεί με τη χρήση τους. Οι πύλες μπορούν να παρασταθούν ως γραμμικοί πίνακες ή γραμμικοί τελεστές. Για παράδειγμα η πύλη NOT φαίνεται παρακάτω.

Είσοδοι 0 και 1 αντίστοιχα: $|0>=\begin{bmatrix}1\\0\end{bmatrix}$, $|1>=\begin{bmatrix}0\\1\end{bmatrix}$ [13]

Πύλη NOT: $N=\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$

Με είσοδο 1 παίρνουμε έξοδο 0:

$N|1>=\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}=\begin{bmatrix}1\\0\end{bmatrix}=|0>$

4.5 Από τη μηχανή Turing στον κβαντικό υπολογιστή

Ο κβαντικός υπολογιστής δε θα επεξεργάζεται bits αλλά Qubits [13],[14].

Το qubit είναι το βασικό σημείο αποθήκευσης κβαντικής πληροφορίας. Έχει δυο επιτρεπτές καταστάσεις, την |1> και την |0>.

Πάρα ταύτα μπορεί κάθε φορά να βρίσκεται σε υπέρθεση (superposition) η οποία αποτελεί τη λυδία λίθο και βασική έννοια των κβαντικών υπολογιστών. Δηλαδή μπορεί κάθε στιγμή να έχει τιμές :

|ψ>=α |0> +β|1>

με $|\alpha|^2+|\beta|^2=1$

$0\leq|\alpha|^2\leq1$ και $0\leq|\beta|^2\leq1$

Και εκφράζεται με ένα πίνακα της μορφής |ψ>= $\begin{bmatrix}\alpha\\\beta\end{bmatrix}$. Ουσιαστικά, θα μπορούσαμε να πούμε ότι ένα qubit μπορεί να βρίσκεται ταυτόχρονα και στις δύο καταστάσεις.

Μπορούμε να επεξεργαστούμε τα qubit κατ' αναλογία με τα bit στις λογικές πύλες, όπως φαίνεται στη συνέχεια:

2 λογικές γραμμές φέρουν τα qubit ψ1,ψ2 :

$$|\psi_1 >= \begin{bmatrix}\alpha_1\\ \beta_1\end{bmatrix}, |\psi_2 >= \begin{bmatrix}\alpha_2\\ \beta_2\end{bmatrix}$$

Μπορούν να αναπαρασταθούν με το γινόμενό τους:

$$|\psi_1 > \otimes |\psi_2 >= |\psi_1\psi_2 >$$

Ο 2x2 πίνακας γίνεται ένα διάνυσμα 4x2 και περιέχει όλες τις πιθανές καταστάσεις των qubit:

$$|\psi_1\psi_2 >= \begin{bmatrix}\alpha_1 & \alpha_2\\ \alpha_1 & \beta_2\\ \beta_1 & \alpha_2\\ \beta_1 & \beta_2\end{bmatrix}$$

Με ανάλογο συλλογισμό, εάν έχουμε 3 qubit τότε στην ουσία έχουμε κατ' αναλογία όλες τις πιθανές καταστάσεις τους που είναι 8

000, 001, 010, 011, 100, 101, 110, 111

Άρα πολύ περισσότερη πληροφορία από το ενδεχόμενο να έχουμε απλά bits.

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

Οι χρήσεις των κβαντικών υπολογιστών έχουν απεριόριστες δυνατότητες σε κάποια πεδία της επιστήμης όπως θα δούμε στη συνέχεια. Ωστόσο, δεν εφαρμόζονται και δεν προσφέρουν ταχύτητα και ευκολία σε όλα τα μαθηματικά προβλήματα αλλά σε κάποια εξ αυτών.

Στην κρυπτογραφία συγκεκριμένα, πολλοί αλγόριθμοι στηρίζονται στο μαθηματικό πρόβλημα της παραγοντοποίησης μεγάλων αριθμών. Ένας από αυτούς τους κρυπτογραφικούς αλγορίθμους όπως τον μελετήσαμε, είναι και ο RSA. Με την υπάρχουσα τεχνολογία θεωρείται ασφαλής, αλλά με την έλευση του κβαντικού υπολογιστή αυτό δεν συμβαίνει. Ειδικότερα, ο αλγόριθμος RSA καταρρέει υπό το βάρος του αλγορίθμου Shor – εφαρμοσμένο σε κβαντικό υπολογιστή.

4.6 Ο αλγόριθμος Shor

Ο τρόπος λειτουργίας των κβαντικών υπολογιστών παρέχει τη δυνατότητα δημιουργίας ειδικών αλγορίθμων, όπως ο αλγόριθμος του Shor αλλά και άλλων, που υλοποιούνται σε κβαντικό υπολογιστή. Έτσι εκμεταλλεύονται τις ευεργετικές δυνατότητες του κβαντικού υπολογιστή και μπορούν να επιλύουν μαθηματικά ζητήματα σε ρεαλιστικό πεπερασμένο χρόνο, κάτι που θα έμοιαζε αδύνατο σε ένα συμβατικό υπολογιστή. Στα εργαστήρια της AT&T το 1994 ο Peter Shor πρότεινε τον αλγόριθμο που φέρει το όνομά του και επιλύει το γνωστό πρόβλημα της παραγοντοποίησης μεγάλων αριθμών σε πολυωνυμικό χρόνο[18]. Αυτό φυσικά έχει τις γνωστές συνέπειες στο κρυπτοσυστημα RSA. Παρουσιάζουμε παρακάτω τα βήματα του αλγορίθμου αυτού:

i. Έστω δοθείς αριθμός N προς παραγοντοποίηση.

ii. Επιλέγεται ένας τυχαίος αριθμός α < N.

iii. Υπολογίζεται ο μέγιστος κοινός διαιρέτης gcd(a, N) με χρήση του αλγορίθμου του Ευκλείδη.

iv. Αν βρεθεί ο gcd(a, N) και είναι διαφορετικός του 1 τότε υπάρχει διαιρέτης του N και σταματάμε. Αλλιώς προχωράμε στο βήμα (v).

v. Υπολογισμός περιόδου r συνάρτησης f(x) = a^x modN η οποία υπολογίζεται σε πεπερασμένο πολυωνυμικό χρόνο με τη χρήση ενός μοντέλου υπολογισμών σχεδιασμένο για κβαντικό υπολογιστή.

vi. Αν ο r περιττός ή $r/2 \equiv -1 \pmod N$, επιστροφή στο βήμα (ii)

vii. Οι παράγοντες του N είναι οι gcd($a^{r/2}$ + 1, N) και gcd($a^{r/2}$ - 1, N)

Ο αλγόριθμος αυτός όπως έχει σχεδιαστεί, δίνει τη δυνατότητα, με κάποιες διαφοροποιήσεις, να επιτευχθεί μεγάλη βελτίωση και σε άλλα δύσκολα προβλήματα, όπως το πρόβλημα διακριτού λογαρίθμου (DLOG) αλλά και σε ζητήματα ελλειπτικών καμπυλών.

Σχετικά με τον AES, αλλά και γενικότερα σχετικά με την κρυπτογραφία συμμετρικού κλειδιού, έχουμε ήδη τον αλγόριθμο Grover ο οποίος έχει σχεδιαστεί για κβαντικό υπολογιστή και μεταβάλλει το χρόνο εξαντλητικής αναζήτησης σε πολυωνυμικό. Βέβαια με την αύξηση μεγέθους κλειδιού η επεξεργασία δε γίνεται σε ρεαλιστικό πεπερασμένο χρόνο και έτσι δεν δύναται να σπάσει ο αλγόριθμος το κλειδί μιας ασφαλούς συνεδρίας με ένα κλειδί AES. Προς το παρόν λοιπόν η αύξηση στα bit του μυστικού κλειδιού στο AES μας καλύπτει ως προς την ασφάλεια στην περίπτωση έλευσης κβαντικού υπολογιστή. Γενικά όμως καταλαβαίνουμε ότι η κατασκευή ενός κβαντικού υπολογιστή θα έχει άμεσες συνέπειες σε κρυπτοσυστήματα τα οποία θα είναι πλέον ευάλωτα.

Name	**function**	**pre-quantum security level**	**post-quantum security level**
Symmetric cryptography			
AES-128 [1]	block cipher	128	64 (Grover)
AES-256 [1]	block cipher	256	128 (Grover)
Salsa20 [2]	stream cipher	256	128 (Grover)
GMAC [3]	MAC	128	128 (no impact)
Poly1305 [4]	MAC	128	128 (no impact)
SHA-256 [5]	hash function	256	128 (Grover)
SHA-3 [6]	hash function	256	128 (Grover)
Public-key cryptography			
RSA-3072 [7]	encryption	128	broken (Shor)
RSA-3072 [7]	signature	128	broken (Shor)
DH-3072 [8]	key exchange	128	broken (Shor)
DSA-3072 [9, 10]	signature	128	broken (Shor)
256-bit ECDH [11, 12, 13]	key exchange	128	broken (Shor)
256-bit ECDSA [14, 15]	signature	128	broken (Shor)

Εικόνα 4.3: Παρούσα κατάσταση στην ασφάλεια των κλασσικών κρυπτοσυστημάτων σε σχέση με την έλευση του κβαντικού υπολογιστή[06].

Για αυτό το λόγο καταλήγουμε σε δυο κυρίες διεξόδους.

Κβαντική κρυπτογραφία (Quantum Cryptography), που είναι ο κλάδος εκείνος της κρυπτογραφίας που εκμεταλλεύεται τις ιδιότητες του κβαντικού υπολογιστή και αντιμετωπίζει επιθέσεις κρυπτανάλυσης από αυτές τις μηχανές.

Η μετακβαντική κρυπτογραφία (post-quantum Cryptography) ορίζεται ο κλάδος της κρυπτογραφίας που σχεδιάζει αλγορίθμους και κρυπτοσυστήματα τα οποία είναι ανθεκτικά στις

επιθέσεις με κβαντικό υπολογιστή(quantum proof). Τέτοια κρυπτοσυστήματα βασίζουν την ασφάλειά τους είτε σε ιδιότητες των πλεγμάτων (lattices), είτε σε συστήματα τετραγωνικών συναρτήσεων πολλών μεταβλητών είτε σε συστήματα διόρθωσης σφαλμάτων και τη θεωρία κωδίκων. όπως το κρυπτοσυστημα McEliece που θα μελετηθεί εκτενέστερα στην παρούσα εργασία.

Η μετακβαντική κρυπτογραφία λοιπόν είναι τελείως διαφορετική από τη κβαντική κρυπτογραφία. Η κβαντική κρυπτογραφία βασίζεται σε ιδιότητες της κβαντομηχανικής για τη δημιουργία ενός κλειδιού κρυπτογράφησης. Μάλιστα, το κύριο χαρακτηριστικό της κβαντικής κρυπτογραφίας, το οποίο τη διαφοροποιεί από τη συμβατική κρυπτογραφία, είναι το ότι κάθε παρατήρηση του κλειδιού από κάποιον υποκλοπέα γίνεται αμέσως αντιληπτή, γιατί κάθε παρατήρηση (μέτρηση) διαταράσσει το κβαντικό σύστημα. Η μετακβαντική κρυπτογραφία – στην οποία εστιάζει η παρούσα διατριβή - δεν είναι τίποτα άλλο από τη συμβατική κρυπτογραφία, η οποία όμως παρέχει ασφάλεια ακόμα και αν θεωρηθεί ότι είναι διαθέσιμοι κβαντικοί υπολογιστές – ήτοι ότι οι αλγόριθμοι των Shor και Grover μπορούν να εκτελεστούν σε παρόντα χρόνο.

Κεφάλαιο 5
Μετακβαντική κρυπτογραφία

5.1 Ιστορική αναδρομή

Σε μερικά χρόνια από τώρα είναι πιθανόν κάποια εταιρία να ανακοινώσει την επιτυχή εμπορική υλοποίηση και διάθεση ενός κβαντικού υπολογιστή καθώς οι προσπάθειες για την κατασκευή του είναι σε εξέλιξη. Το πιθανότερο είναι ότι αμέσως θα ξεσπάσει μία ιλιγγιώδης διαδικασία αμφισβήτησης πολλών γνωστών αλγορίθμων κρυπτογραφίας δημοσίου κλειδιού. Αυτό προκύπτει από την προαναφερθείσα, σε αυτή την εργασία, θέση πως ο κβαντικός υπολογιστής μπορεί να «υπερνικήσει» γνωστές κρυπτογραφικές μεθόδους δημόσιου κλειδιού. Το άμεσο πρόβλημα έγκειται στο γεγονός ότι στις περισσότερες σημερινές εφαρμογές στο ίντερνετ χρησιμοποιούνται ευρέως αλγόριθμοι δημοσίου κλειδιού, όπως ο RSA και αλγόριθμοι ελλειπτικών καμπυλών.

Βεβαίως, δεν είναι αυτά τα μόνα κρυπτογραφικά συστήματα δημοσίου κλειδιού τα οποία είναι γνωστά μέχρι στιγμής. Έχουν προταθεί από επιστήμονες αλγόριθμοι ως προς την αντοχή τους στους κβαντικούς υπολογιστές και τα συναφή κρυπτοσυστήματα. Υπάρχουν τα συστήματα τα βασισμένα στην θεωρία κωδίκων, στη θεωρία πλέγματος αλλά και πολλές άλλες υλοποιήσεις στην επιστημονική κοινότητα. Με λίγα λόγια υπάρχουν κρυπτοσυστήματα τα οποία πιστεύεται ότι θα αντισταθούν επιτυχώς στις δυνατότητες του κβαντικού υπολογιστή. Οι αλγόριθμοι Grover και Shor παίζουν σημαντικό ρόλο, όπως προαναφέραμε, στην κατάρρευση μιας οικογένειας από κρυπτοσυστήματα τα οποία βασίζονται στην παραγοντοποίηση μεγάλων αριθμών.

Οι επιστήμονες έχουν αφιερώσει πολύ μεγάλο χρονικό διάστημα στην κρυπτανάλυση διότι με την εξέλιξη της κρυπτανάλυσης μπορεί να διαφανεί κατά πόσον ένας αλγόριθμος ό οποίος χρησιμοποιείται σήμερα θα πάψει να είναι ασφαλής στο μέλλον. Τις περισσότερες φορές αυτό που συμβαίνει είναι ότι ο αλγόριθμος πραγματικά «καταρρίπτεται» από κάποια ειδική μέθοδο κρυπτανάλυσης. Όμως σε κάποιες περιπτώσεις η ασφάλεια καταρρίπτεται λόγω του μεγέθους του κλειδιού – το οποίο μπορεί να είναι αρκετά μεγάλο με τα εκάστοτε τρέχοντα τεχνολογικά δεδομένα αλλά όχι αρκούντως μεγάλο στο μέλλον – οπότε και το ζήτημα αυτό αντιμετωπίζεται με αύξηση του μεγέθους κλειδιού. Κάτι τέτοιο ισχύει και για την επίδραση που θα έχουν οι κβαντικοί υπολογιστές στον αλγόριθμο AES. Αυτό όμως δεν ισχύει, όσον αφορά την επίδραση που θα έχουν οι κβαντικοί υπολογιστές, σε κάποιους αλγόριθμους δημοσίου κλειδιού.

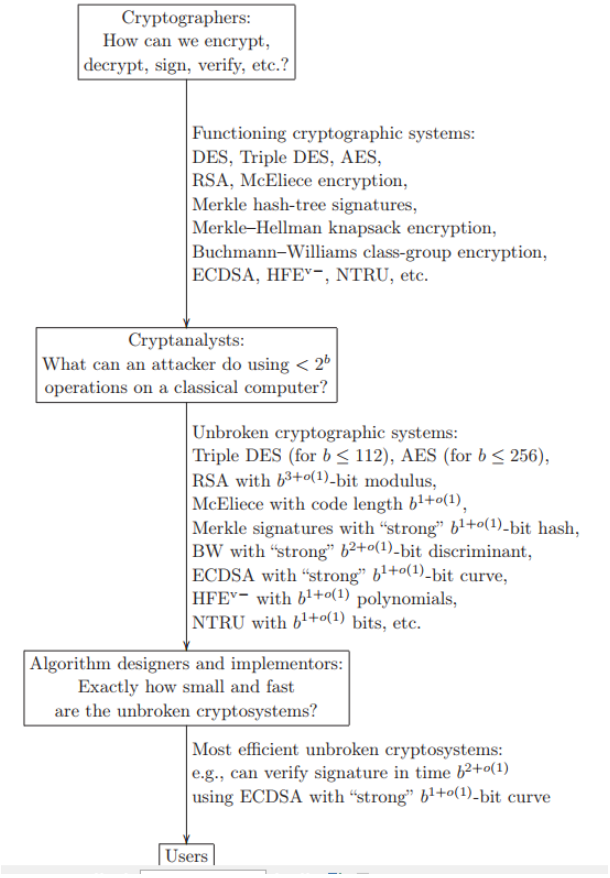
Γενικότερα, όσο περισσότερα χρόνια ένα υπάρχον κρυπτοσύστημα παραμένει ανθεκτικό σε κρυπταναλυτικές επιθέσεις τόσο περισσότερο εδραιώνεται στην διεθνή κοινότητα αλλά και εμπορικά στο διαδίκτυο. Ένα παράδειγμα είναι ο αλγόριθμος δημοσίου κλειδιού RSA. Γενικότερα η διαδικασία σχεδίασης ανάλυσης και καλυτέρευσης των κρυπτογραφικών συστημάτων πριν το τετελεσμένο του κβαντικού υπολογιστή έχει ως εξής:

i. Οι μαθηματικοί και οι κρυπτογράφοι σχεδιάζουν συστήματα για την κρυπτογράφηση και αποκρυπτογράφηση δεδομένων.

ii. Ειδικοί στην κρυπτανάλυση καταφέρνουν να «καταρρίψουν» κάποια από τα συστήματα που έχουν σχεδιαστεί στο προηγούμενο βήμα.

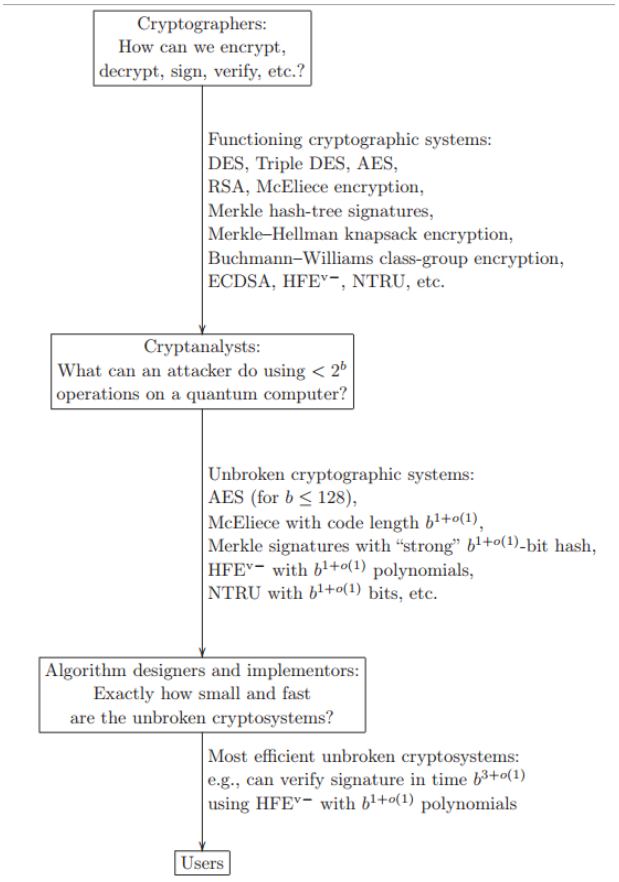
iii. Σχεδιαστές αλγορίθμων μελετούν και αποτιμούν την ταχύτητα και τις ιδιότητες υλοποίησης των αλγορίθμων που είναι ανθεκτικοί στις επιθέσεις κρυπτανάλυσης.

Παρακάτω φαίνεται σχηματικά η ιστορία της προ-κβαντικής κλασικής κρυπτογραφίας με βάση τη διαδικασία που αναφέρθηκε παραπάνω.



Εικόνα 5.1: Διαδικασία σχεδίασης, επανασχεδίασης αλγορίθμων στην κλασσική κρυπτογραφία [04]

ενώ στο επόμενο σχήμα φαίνεται αντίστοιχα η σχεδίαση και η απόκριση των κρυπταναλυτών με τη χρήση και κβαντικών αλγορίθμων και την θεωρητική ύπαρξη κβαντικού υπολογιστή



Εικόνα 5.2: Διαδικασία σχεδίασης και επανασχεδίασης αλγορίθμων στην μετακβαντική κρυπτογραφία [04].

Γίνεται πλέον αντιληπτό ότι η μετακβαντική κρυπτογραφία θα βοηθήσει πολύ σε ένα ασφαλές διαδικτυακό μέλλον καθώς και στην ομαλή μετάβαση από αλγόριθμους κλασικής κρυπτογραφίας σε αλγόριθμους μετακβαντικής κρυπτογραφίας. Κατά την μελέτη πάνω στην μετακβαντική κρυπτογραφία κυρίως εστιάζουμε στα κρυπτοσυστήματα Δημόσιου κλειδιού λόγω του αλγορίθμου Shor. Ο αλγόριθμος Grover «επιφέρει» την ανάγκη αύξησης του μεγέθους του κλειδιού στους συμμετρικούς αλγορίθμους. ο αλγόριθμος του Shor όμως «καταρρίπτει» πλήρως την ασφάλεια σε κάποιους αλγορίθμους δημοσίου κλειδιού.

5.2 Βιομηχανικά προϊόντα με μεγάλο κύκλο ζωής

Σε αυτό το σημείο θα πρέπει να επισημανθεί το γεγονός ότι σε κάποιους τομείς τις βιομηχανίας τα προϊόντα έχουν αρκετά ευρεία διάρκεια χρήσης και υποστήριξης. Για παράδειγμα ένα αυτοκίνητο που πωλείται σήμερα ή έστω σε 5-6 χρόνια από τώρα θα έχει μέσο όρο ζωής ενδεχομένως 15-20

έτη. Οπότε τέτοιου είδους βιομηχανίες πρέπει να έχουν υπόψιν την ευπάθεια των προϊόντων τους απέναντι στην έλευση του κβαντικού υπολογιστή. Όσο καθυστερεί η μελέτη και η εφαρμογή μετακβαντικής κρυπτογραφίας και εν γένει μετακβαντικής τεχνολογίας, τόσο κάποια προϊόντα που πωλούνται σήμερα θα είναι και πιο ευπαθή στο μέλλον. Ήδη κάποιες βιομηχανίες δείχνουν ενδιαφέρον στον τομέα της προσφοράς μετακβαντικών τεχνολογιών ασφάλειας. Η Google ήδη φέρεται να πειραματίζεται με ένα μετακβαντικό κρυπτοσύστημα δημοσίου κλειδιού, το NewHope [01] για την επικοινωνία μεταξύ του browser Chrome και κεντρικών Servers της. Και η INTEL αντίστοιχα εργάζεται πάνω στην υλοποίηση μετακβαντικής κρυπτογραφίας στα προϊόντα της καθώς και στην διαδικασία παραγωγής τους, όπως για παράδειγμα η ασφαλής επικοινωνία των εργοστασίων μεταξύ τους. Υπάρχουν μάλιστα υλοποιημένα μετακβαντικά κρυπτοσυστήματα που είχαν διατεθεί εμπορικά από το 1990 (NTRUSign & NTRUencrypt) από την security Innovation. Πολλές ακόμη εταιρείες διαθέτουν βιβλιοθήκες λογισμικού που περιλαμβάνουν μετακβαντικούς αλγορίθμους όπως για παράδειγμα το Department of Computer Science at Technische Universität Darmstadt, Germany. Το πανεπιστήμιο αυτό «τρέχει» το Project Flexiprovider που συμπεριλαμβάνει πολύ καλές βιβλιοθήκες java για μετακβαντικούς αλγορίθμους, όπως ο McEliece που χρησιμοποιήθηκε στην υλοποίηση ενός μετακβαντικού μοντέλου chat στην παρούσα εργασία. Όλα τα παραπάνω καταδεικνύουν το πέρασμα με σταθερά βήματα της μετακβαντικής κρυπτογραφίας, πέρα από το ακαδημαϊκό πλαίσιο, στον πραγματικό κόσμο [18].

5.3 Μετάβαση σε μετακβαντικό περιβάλλον-ζητήματα που προκύπτουν

Φυσικά υπάρχει ένας μεγάλος αριθμός από εμπόδια που πρέπει να προσπεραστούν για να ενεργοποιηθεί η μετακβαντική κρυπτογραφία στην πράξη και στην καθημερινότητα. Ένα από τα κυριότερα είναι η εμπιστοσύνη του κοινού, των ενδιαφερομένων μερών [18]. Στην πράξη έχει αποδειχθεί ότι όσο χρονικό διάστημα ένας κρυπτογραφικός αλγόριθμος παραμένει απαραβίαστος τόσο περισσότερη εμπιστοσύνη υπάρχει προς αυτόν. Έτσι, πολλά κρυπτοσυστήματα μετακβαντικής κρυπτογραφίας είναι γνωστά για αρκετά χρόνια χωρίς να έχουν δεχθεί κρίσιμες επιθέσεις, και άρα μπορούν να υλοποιηθούν πρακτικά. Άλλα κρυπτοσυστήματα αντιθέτως είναι πολύ καινούργια. Από αυτή την παραδοχή κατανοούμε ότι δεν έχουμε και πολύ χρόνο μπροστά μας ώστε τα νέα κρυπτοσυστήματα να τύχουν αποδοχής και

εμπιστοσύνης. Δεν υπάρχει χρονικό περιθώριο καθώς ήρθε η στιγμή που ειπώθηκε ότι ο κβαντικός υπολογιστής είναι υπό κατασκευή [20]. Έτσι, θα πρέπει η μετακβαντική κρυπτογραφία να εστιάσει περισσότερο στο ενδελεχές τεστ των νέων κρυπταλγορίθμων παρά απλά να περιμένει η πανεπιστημιακή κοινότητα να περάσει ο χρόνος και έτσι να γίνουν ώριμα, αρά και αποδεκτά και ασφαλή, τα κρυπτοσυστήματα αυτά.

Οι μετρικές (metrics) που χρησιμοποιούνται αναφορικά με την αποτίμηση των κλασσικών επιθέσεων έναντι κρυπτοσυστημάτων αποτελούν ένα άλλο σημείο που πρέπει να είναι σαφώς ορισμένο. Οι παράμετροι ασφαλείας πρέπει να επιλεχθούν έτσι ώστε η πιο επικίνδυνη επίθεση κρυπτανάλυσης να έχει ένα κόστος ισχύος μηχανών, αρκούντως μεγάλο. Παρόλα αυτά δεν είναι σαφώς ορισμένο ούτε το κόστος αυτό αλλά ούτε και το κατώφλι πέρα από το οποίο κάποιο κρυπτοσύστημα μπορεί να χαρακτηριστεί ασφαλές. Ο λόγος είναι ότι ανακαλύπτονται διαρκώς ολοένα και ισχυρότερες κρυπταναλυτικές τεχνικές αλλά και συνεχώς βελτιούμενα υπολογιστικά συστήματα συμβατικής τεχνολογίας. Άρα οι τιμές των παραμέτρων ασφαλείας των κρυπτογραφικών αλγορίθμων πρέπει να ανανεώνονται συνεχώς. Με τον ίδιο τρόπο πρέπει να «αντιδράσει» και η μετακβαντική κρυπτογραφία. Πρέπει λοιπόν, και στους αλγορίθμους μετακβαντικής κρυπτογραφίας, να ανανεώνονται οι μετρικές και οι ελάχιστες επιτρεπτές τιμές παραμέτρων που απαιτούνται για να παραμένουν ασφαλείς οι μέθοδοι και τα κρυπτοσυστήματα απέναντι σε επιθέσεις, με ή χωρίς την υπόθεση ύπαρξης κβαντικού υπολογιστή. Αυτό προσκρούει μερικώς στην μη εκτεταμένη γνώση των συνολικών δυνατοτήτων του κβαντικού υπολογιστή. Φυσικά είναι κάτι που το περιμένουμε αφού δεν υπάρχει ακόμη διαθέσιμη συσκευή, προτυποποιημένη, ώστε να έχουμε ένα πρακτικό μοντέλο κβαντικού υπολογιστή που να μετρούνται και να δοκιμάζονται οι πιθανές επιθέσεις. Για αυτό το λόγο όλες οι προσδοκίες και οι ορισμοί για την ασφάλεια των μετακβαντικών συστημάτων βρίσκονται αμιγώς σε θεωρητικό επίπεδο. Ίσως λοιπόν, η δυνατότητα του κβαντικού υπολογιστή είναι υποτιμημένη και οι παράμετροι ασφάλειας που διαφαίνονται να αποδίδουν θεωρητικά να μην ισχύουν στην πράξη, οπότε και τα φερόμενα ανθεκτικά ως προς τους κβαντικούς υπολογιστές (quantum proof) κρυπτοσυστήματα τελικά να μην είναι ανθεκτικά. Από την άλλη πλευρά, αν οι δυνατότητες του κβαντικού υπολογιστή είναι υπερεκτιμημένες, τότε η επιλογή παραμέτρων ασφάλειας των μετακβαντικών κρυπτοσυστημάτων δε θα είναι η βέλτιστη ως προς την ευχρηστία τους και αυτό ενδέχεται να επηρεάσει δυσμενώς την απόδοσή τους, την ευκολία υλοποίησής τους και, εν τέλει, την καθολικότητά τους κυρίως σε προτυποποιημένα πρωτόκολλα για ευρεία χρήση στο Διαδίκτυο. Για αυτό το λόγο πρέπει άμεσα να βρεθεί η σωστή «μετρική» για την αξιολόγηση των μετακβαντικών αλγορίθμων, όπως έχει γίνει και για τους κλασσικούς κρυπταλγόριθμους [18].

Όταν βρεθεί μία συγκεκριμένη μετρική για τους μετακβαντικούς αλγόριθμους θα πρέπει να προσδιορίσουμε τις παραμέτρους ασφαλείας για αυτά τα συστήματα. Προς το παρόν υπάρχουν πολλά άρθρα που επικεντρώνονται στις παραμέτρους ασφαλείας που καθιστούν τα κρυπτοσυστήματα ασφαλή έναντι των κλασικών απειλών, επειδή αυτό διευκολύνει την σύγκριση με τα κλασσικά κρυπτοσυστήματα, αφού οι μετρικές που χρησιμοποιούνται στην συμβατική κρυπτογραφία είναι καλώς και επαρκώς κατανοητές. Όμως το κύριο πλεονέκτημα των μετακβαντικών συστημάτων κρυπτογραφίας είναι η ανθεκτικότητά τους τους στον κβαντικό υπολογιστή και στις επιθέσεις του. Γι' αυτό πρέπει να υπάρξουν μετακβαντικές παράμετροι ασφαλείας για τα αντίστοιχα συστήματα ασφαλείας.

Άλλο ένα θέμα το οποίο εξετάζεται στην μετακβαντική κρυπτογραφία είναι η αποδοτικότητα των κρυπτοσυστημάτων, όπου παρατηρούνται μεγάλες διαφορές μεταξύ των κρυπτοσυστημάτων. Πρέπει να τονίσουμε ότι δεν υπάρχουν πολλά μετακβαντικά κρυπτοσυστήματα που να έχουν καλή αποδοτικότητα όταν αυτά συγκριθούν με κλασσικά. Γι' αυτό το λόγο θέλουμε να υπάρχουν βελτιώσεις για τα μετακβαντικά κρυπτοσυστήματα ώστε να μειώσουμε τους υπολογιστικούς πόρους κατά την χρήση τους, οι οποίοι προς το παρόν φαίνονται να είναι αρκετά αυξημένοι. Υπάρχει έρευνα ακόμα και για την ελάττωση του μεγέθους κλειδιού των μετακβαντικών συστημάτων κρυπτογραφίας. Ως προς την προσπάθεια μείωσης των υπολογιστικών πόρων έχουν γίνει κάποια βήματα, αλλά έχουν αποτύχει διότι είχαν ως αποτέλεσμα την απώλεια στην ασφάλεια[18]. Το τίμημα μακροβιότερης ασφαλούς μετακβαντικής κρυπτογραφίας είναι το μεγαλύτερο κόστος σε υπολογιστική ισχύ αποθήκευσης αλλά και η χωρητικότητα καναλιών επικοινωνίας. Στην ουσία τα επόμενα χρόνια θα υπάρξουν μεγάλες προσπάθειες για την επίτευξη καλύτερης αποδοτικότητας στους υπάρχοντες μετακβαντικούς αλγόριθμος.

Μια άλλη πραγματικότητα είναι ότι οι συσκευές οι οποίες παρέχουν υπολογιστική ισχύ που θα μπορούσε να «αντέξει» τους αλγόριθμους της μετακβαντικής κρυπτογραφίας κοστίζουν αρκετά. Κάποιες συμβατικές συσκευές που ήδη λειτουργούν, δε θα είναι ικανές να υλοποιήσουν αλγόριθμους μετακβαντικής κρυπτογραφίας, οπότε σίγουρα θα χρειαστεί δαπάνη του τελικού χρήστη για νέα μηχανήματα.

Σε σχέση με την ασφαλή υλοποίηση και εξασφάλιση των μετακβαντικών αλγορίθμων υπάρχουν νέα θέματα που θα μας απασχολήσουν στο μέλλον. Ήδη στην κλασική κρυπτογραφία η κρυπτανάλυση παράπλευρου καναλιού (side channel cryptanalysis) αλλά και γενικότερα οι τεχνικές κρυπτανάλυσης που βασίζονται σε μετρήσεις φυσικών ποσοτήτων (physical cryptanalysis) έχουν γίνει πολύ μεγάλες απειλές για συμβατικούς κρυπταλγόριθμους, αλλά πρέπει

να τεθούν υπόψη μας για την αντιμετώπιση παρόμοιων απειλών σε μετακβαντικά κρυπτογραφικά σχήματα. Επίσης θα πρέπει να περιμένουμε και νέα είδη απειλών που ενδεχομένως να βασίζονται στον κβαντικό υπολογιστή και τις δυνατότητές του, που ακόμα δεν έχουμε εξερευνήσει πλήρως για λόγους που προαναφέρθηκαν.

Ένα ακόμα σημαντικό ζήτημα είναι η μετακίνηση των συστημάτων από το περιβάλλον της συμβατικής κρυπτογραφίας στο περιβάλλον της μετακβαντικής κρυπτογραφίας. Κατά την περίοδο αυτή της εναλλαγής θα πρέπει να υπάρχει ένα υβριδικό μοντέλο που να χρησιμοποιηθεί και τους δύο αλγόριθμους και αυτό φαίνεται ότι θα πρέπει να είναι υποχρεωτικό. Άλλωστε η εταιρεία Google χρησιμοποιεί αυτήν την προσέγγιση σε ερευνητικό επίπεδο με τη χρήση του σχήματος NewHope[01].

Οι ενημερώσεις ασφάλειας στους μετακβαντικούς αλγόριθμους πρέπει να γίνεται σε επίπεδο λογισμικού (software) και, άρα, πολλές συσκευές όπως τα RFID θα μπορούσαν ήδη να καταργηθούν κατά την προετοιμασία για την έλευση του κβαντικού υπολογιστή και της διττής μορφής – μοντέλου υβριδικής κρυπτογραφίας στο Διαδίκτυο. Αυτό μας θυμίζει και την εφαρμογή του υβριδικού Μοντέλου για το IPv6 και το IPv4 με τους δρομολογητές Dual Stack. Πάντα θα χρειάζεται μια ομαλή υβριδική περίοδος μετάβασης σε τεχνολογίες τόσο μεγάλου βεληνεκούς.

Άλλο ένα ζήτημα που ανακύπτει, αφορά στο γεγονός ότι αντικειμενικά δε χρειάζεται να χρησιμοποιούμε τώρα μετακβαντική κρυπτογραφία δημόσιου κλειδιού αφού δεν υπάρχει ο κβαντικός υπολογιστής. Οπότε είμαστε ασφαλείς με την κλασική συμβατική κρυπτογραφία. Εκεί που υπάρχει πρόβλημα είναι σε αποθηκευμένα δεδομένα κρυπτογραφημένα με μέθοδο συμμετρικού κλειδιού κλασικής κρυπτογραφίας και το ενδεχόμενο τα δεδομένα αυτά να είναι διαθέσιμα σε μετακβαντικό χρόνο – δηλαδή σε χρόνο κατά τον οποίο θα είναι πραγματικότητα οι κβαντικοί υπολογιστές. Τότε, κάποιος επιτιθέμενος μπορεί να τα έχει στη διάθεσή του για να τα επεξεργαστεί με κβαντικό Υπολογιστή, αρά να καταρρίψει offline τα κλασσικά πρωτόκολλα και να αποκομίσει πληροφορίες άκρως απόρρητες.

Υπάρχουν πάρα πολλά προτυποποιημένα συστήματα υπογραφών και προστασίας προσωπικών δεδομένων τα οποία δεν κατευθύνουν τον αναλυτή στην επιλογή κρυπτοσυστημάτων, αλλά απλά υπερθεματίζουν την αδιάλειπτη προφύλαξη των δεδομένων έναντι διαφόρων κρυπταναλυτικών επιθέσεων. Και σε αυτό το επίπεδο θα χρειαστεί εισαγωγή της νέας τεχνολογίας μετακβαντικής κρυπτογραφίας για να προστατευτούν τα δεδομένα, σύμφωνα και με τις νομικές απαιτήσεις (για παράδειγμα, στο Γενικό Κανονισμό (ΕΕ) 2016/679 για την προστασία προσωπικών δεδομένων,

γίνεται ρητή αναφορά στο ότι τόσο για την επιλογή των κατάλληλων μέτρων ασφάλειας όσο και για την αξιολόγηση των κινδύνων θα πρέπει να λαμβάνονται υπόψη και οι τελευταίες τεχνολογικές εξελίξεις).

Θα πρέπει να τονίσουμε ότι πολλές φορές οι κβαντικοί υπολογιστές κακώς πιστεύεται ότι παρέχουν λύσεις άμεσα σε οποιοδήποτε υπολογιστικό πρόβλημα. Όπως έχουμε ξαναπεί στην παρούσα εργασία αυτή δεν είναι η πραγματικότητα. Η πραγματική δύναμη των κβαντικών υπολογιστών περιορίζεται σε ειδικούς αλγόριθμους, και άρα σε ένα εύρος εφαρμογών. Η κρυπτανάλυση είναι μία από τις εφαρμογές στις οποίες θα συντελέσει θετικά η έλευση του κβαντικού υπολογιστή και έτσι ο αντίκτυπος στο πεδίο των ασφαλών επικοινωνιών είναι σοβαρότατος. Για να εξάγουμε τα σωστά συμπεράσματα για την υλοποίηση μετακβαντικών τεχνικών και τους νομικούς κανονισμούς θα πρέπει όλοι οι εμπλεκόμενοι φορείς – νομικοί, μηχανικοί κτλ. - να είναι σωστά πληροφορημένοι σε σχέση με τον αντίκτυπο του κβαντικού υπολογιστή και τις λύσεις τις οποίες προσφέρει η κρυπτογραφία.

5.4 NIST PQC Project

Το 2016 ήδη ο διεθνής οργανισμός προτυποποίησης NIST (National Institute of Standards and Technology) ξεκίνησε μία διαδικασία αξιολόγησης με στόχο την επιλογή καθολικών προτύπων αλγορίθμων μετακβαντικής κρυπτογραφίας[21],[22]. Επίσης το ευρωπαϊκό ινστιτούτο τηλεπικοινωνιών ETSI (European Telecommunications Standards Institute) δουλεύει προς την κατεύθυνση της ασφαλούς μετακβαντικής κρυπτογραφίας. Αυτή η διαδικασία σε επίπεδο οργανισμών εξαρτάται πολύ σημαντικά από τα πανεπιστήμια και την συμμετοχή τους σε πρωτοπόρες διαδικασίες ως προς την έρευνα στον τομέα της μετακβαντικής κρυπτογραφίας. Τη στιγμή που γράφεται η εργασία είναι σε εξέλιξη ο πρώτος γύρος της διαδικασίας που εκκίνησε ο NIST. Έχουν υποβληθεί 70 προτάσεις με συγκεκριμένα standards αλλά και πηγαίο κώδικα και μετρήσεις, ενώ 5 προτάσεις έχουν αποσυρθεί. Προηγήθηκε ένα συνέδριο που αποφάσισε για τα χαρακτηριστικά που θα πρέπει να έχουν οι αλγόριθμοι που θα υποβληθούν στον πρώτο γύρο προς αξιολόγηση. Αναμένεται σύντομα να ξεκινήσει ο $2^{ος}$ γύρος διαβουλεύσεων σχετικά με τις προτάσεις που πέρασαν επιτυχώς το $1^{ο}$ γύρο. Περίπου το 2021 αναμένεται να αρχίζουν να διαφαίνονται οι επικρατέστερες προτάσεις αλγορίθμων ενώ το 2024 ο οργανισμός έχει στο χρονοδιάγραμμα του την οριστικοποίηση και συγγραφή των πρώτων προτύπων.

Το ερώτημα για το πότε θα κατασκευαστεί ο κβαντικός υπολογιστής μεγάλης κλίμακας, είναι περίπλοκο. Ενώ στο παρελθόν ήταν ασαφές αν οι κβαντικοί υπολογιστές είναι μέσα στα πλαίσια του επιτεύξιμου, πολλοί επιστήμονες πιστεύουν ότι πρόκειται για μια πρόκληση στον τομέα της μηχανικής των υπολογιστών αλλά και της εφαρμοσμένης κβαντικής φυσικής. Μερικοί εμπειρογνώμονες προβλέπουν ότι μέσα στα επόμενα 20 περίπου χρόνια, θα κατασκευαστούν αρκετά μεγάλα συστήματα κβαντικών υπολογιστών και θα προσβάλουν ουσιαστικά όλα τα κρυπτοσυστήματα δημόσιων κλειδιών που χρησιμοποιούνται σήμερα. Έχουν διατεθεί σχεδόν 20 χρόνια για την ανάπτυξη της σύγχρονης υποδομής κρυπτογραφίας δημόσιου κλειδιού. Θα χρειαστούν σημαντικές προσπάθειες για να διασφαλιστεί η ομαλή και ασφαλής μετάβαση από τα τρέχοντα ευρέως χρησιμοποιούμενα κρυπτοσυστήματα σε αντίστοιχα ανθεκτικά στη μετακβαντική εποχή, κρυπτοσυστήματα. Επομένως, ανεξάρτητα από το αν μπορούμε να υπολογίσουμε την ακριβή ημερομηνία άφιξης της κβαντικής εποχής, πρέπει να ξεκινήσουμε από τώρα να προετοιμάζουμε τα συστήματα ασφαλείας των πληροφοριών μας, για να είμαστε σε θέση να αντιμετωπίσουμε τις προκλήσεις ασφάλειας που ανακύπτουν από την κβαντική υπολογιστική.

Ο NIST έχει ένα μοναδικό ρόλο στην τυποποίηση της μετακβαντικής κρυπτογράφησης, ως μέρος της ευρύτερης αρμοδιότητάς του για την ανάπτυξη προτύπων και κατευθυντήριων γραμμών για την προστασία της ασφάλειας εν γένει σε συστήματα πληροφοριών. Πολλά πρότυπα του NIST, όπως ο AES, έχουν αναπτυχθεί με ευρεία συμμετοχή από ακαδημαϊκούς και βιομηχανικούς φορείς, και έχουν υιοθετηθεί ευρέως επειδή υπήρξαν αποτελεσματικές προτάσεις στην πράξη, βοηθώντας έτσι στην προστασία των πληροφοριακών συστημάτων παγκοσμίως. Ο NIST με την προτυποποίηση της μετακβαντικής κρυπτογραφίας είναι πιθανό να παρέχει παρόμοια οφέλη ως «ασπίδα» για ένα ασφαλές μέλλον. Καθίσταται σαφές ότι η προσπάθεια ανάπτυξης κβαντο-ανθεκτικής τεχνολογίας εντείνεται. Εξίσου σαφές είναι ότι καθίσταται άμεση η ανάγκη να προχωρήσουν οι επενδύσεις για τυποποίηση της νέας αυτής κρυπτογραφίας. Είναι κρίσιμο να ασχοληθεί η παγκόσμια επιστημονική κοινότητα σε σχέση με τα κρυπτογραφικά πρότυπα NIST που πρέπει να υιοθετηθούν από τη βιομηχανία .

5.5 Μαθηματικά προβλήματα που χρησιμοποιούνται στην μετακβαντική κρυπτογραφία

Τα μετακβαντικά κρυπτογραφικά σχήματα στα οποία έχει στραφεί η ερευνητική κοινότητα μπορούν να διακριθούν στις εξής κατηγορίες:

i. κρυπτογραφία που βασίζεται στη θεωρία κωδίκων (Code-based cryptography)

ii. κρυπτογραφία πλέγματος (Lattice-based cryptography)

iii. κρυπτογραφία που βασίζεται σε συναρτήσεις κατακερματισμού(hash functions)

iv. κρυπτογραφία πολλών μεταβλητών(Multivariate cryptography)

v. κρυπτογραφία ελλειπτικών καμπυλών με ιδιαίτερα χαρακτηριστικά (Super singular elliptic-curve isogeny cryptography)

Η μετακβαντική κρυπτογραφία που είναι βασισμένη στη θεωρία κωδίκων και στις συναρτήσεις κατακερματισμού μελετάται από το 1970 και γι' αυτό το λόγο είναι περισσότερο αρεστή, κατανοητή και τυγχάνει εμπιστοσύνης από το ακαδημαϊκό κοινό. Η κρυπτογραφία πολλών μεταβλητών έχει ξεκινήσει το 1980 και το μαθηματικό πρόβλημα στο οποίο βασίζεται είναι καλά κατανοητό. Παρόλα αυτά το να σχεδιάζει κάνεις ένα τέτοιο κρυπτοσύστημα δεν είναι τόσο εύκολο και λίγα τέτοια κρυπτοσυστήματα θεωρούνται ασφαλή. Η Κρυπτογραφία πλέγματος ή δικτυωτών όπως αλλιώς αποκαλείται στην ελληνική βιβλιογραφία, ήρθε στα τέλη του 1990 και χαίρει μίας καλής κριτικής μέχρι στιγμής, ενώ συνεχώς ενισχύεται από τους επιστήμονες. Η κρυπτογραφία η οποία είναι βασισμένη στις ισογενείς ελλειπτικές καμπύλες προτάθηκε το 2006 και επαναπροτάθηκε το 2011 με την τεχνική Super Singular Curves[18]. Ειδικά αυτή η οικογένεια αλγορίθμων είναι πολύ νέα για την μετακβαντική κρυπτογραφία και όχι αρκετά κατανοητή σε μεγάλο βαθμό για πρακτική εφαρμογή .

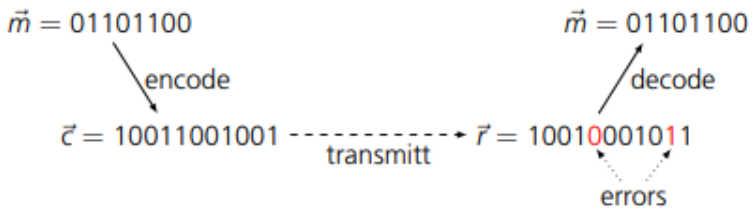
5.5.1 Κρυπτογραφία βασισμένη στη θεωρία κωδίκων

Η βασική ιδέα της κρυπτογραφίας δημοσίου κλειδιού με τη χρήση της θεωρίας κωδίκων είναι η χρήση τεχνικών διόρθωσης λαθών ώστε να κρύψουμε το περιεχόμενο του μηνύματος κατά τη μετάδοση. Στην πραγματικότητα και έως σήμερα οι τεχνικές διόρθωσης λαθών χρησιμοποιούνται ευρέως για την εύρεση λαθών σε επίπεδο bit όταν υπάρχει θόρυβος στο κανάλι επικοινωνίας. Ο κώδικας που θα χρησιμοποιηθεί επιλέγεται σε σχέση με τα επιθυμητά χαρακτηριστικά διόρθωσης που εξαρτώνται από το θόρυβο στο κανάλι επικοινωνίας. Η διορθωτική ικανότητα είναι ένας αριθμός t και εκφράζει τον αριθμό bit που μπορούν να διορθωθούν σε περίπτωση σφάλματος κατά τη μετάδοση. Αρχικά το μήνυμα m μετατρέπεται σε κωδική λέξη c (code word) σύμφωνα με

τον κώδικα που θα χρησιμοποιηθεί. Αυτό προσδίδει την ευχέρεια στην αποκωδικοποίηση και στην εύρεση λαθών καθώς η λέξη που παράγεται είναι μεγαλύτερη του μηνύματος m, κατά τρόπο τέτοιο ώστε τα «πλεονάζοντα» bit να επιτρέπουν διόρθωση σφαλμάτων στον παραλήπτη. Έπειτα η λέξη C μεταδίδεται στο κανάλι. Κατά τη διάρκεια της μετάδοσης πολλά Bits μπορούν να αλλάξουν λόγω του θορύβου. Έτσι ο παραλήπτης δε λαμβάνει το c αλλά το r = c ⊕ e οπού το e είναι ένα διάνυσμα λάθους με κάποιο βάρος (weight) w. Το βάρος w εκφράζει τον αριθμό των bit στο e που είναι 1 , ενώ τα υπόλοιπα είναι 0. Ο παραλήπτης αντιστοιχεί το ληφθέν μήνυμα r στην πιο «κοντινή» λέξη c του κώδικα. Αν το σύνολο των λαθών στο r είναι μικρότερο από τη διορθωτική ικανότητα t του κώδικα, δηλαδή αν $w \leq t$, τότε το c που επέλεξε ο παραλήπτης ως κοντινότερη λέξη στο r είναι πράγματι η σωστή κωδική λέξη c που εστάλη. Έπειτα από αυτή τη διαδικασία ο παραλήπτης εφαρμόζει την αντίστροφη διαδικασία στο επιλεγμένο c και ανακτά το αρχικό κείμενο m. Πρέπει να τονίσουμε ότι η αποκωδικοποίηση κωδίκων απαιτεί μεγάλη υπολογιστική ισχύ και θα μπορούσε να αποτύχει αναλόγως των παραμέτρων του κώδικα που επελέγη. Παρόλα αυτά υπάρχουν κάποιοι κώδικες για τους οποίους έχουν βρεθεί αποδοτικοί αλγόριθμοι αποκωδικοποίησης. Οι τελευταίοι είναι και οι κώδικες που χρησιμοποιούνται στην πράξη. Μία μεγάλη οικογένεια κωδίκων που εμφανίζει αρκετά πλεονεκτήματα ως προς την υλοποίησή της είναι οι γραμμικοί κώδικες (linear codes). Ένας γραμμικός κώδικας έχει την εξής ιδιότητα: το άθροισμα δύο οποιωνδήποτε κωδικών λέξεων είναι επίσης μία κωδική λέξη του κώδικα.

Έστω ένας [n, k, d] γραμμικός κώδικας επί του πεπερασμένου σώματος F.

Ένας k × n πίνακας G, του οποίου οι γραμμές αποτελούν μια βάση του κώδικα (δηλαδή κάθε κωδική λέξη μπορεί να προκύψει από κατάλληλο γραμμικό συνδυασμό των γραμμών του G, ονομάζεται γεννήτορας πίνακας του γραμμικού κώδικα.



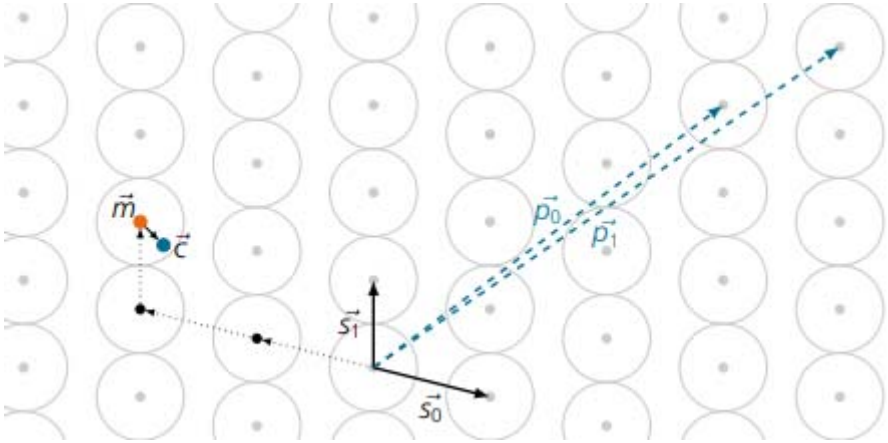
Εικόνα 5.3: Διαδικασία κωδικοποίησης και αποκωδικοποίησης σε συστήματα κωδίκων.

Η διασφάλιση που παρέχεται για τη χρήση τέτοιων μεθόδων στην μετακβαντική κρυπτογραφία είναι ακριβώς η δυσκολία αποκωδικοποίησης ενός τυχαίου γραμμικού κώδικα. Με άλλα λόγια, αν ο κώδικας δεν έχει κάποια συγκεκριμένη, όχι τυχαία, δομή, η αποκωδικοποίησή του δεν είναι εφικτή. Το πρώτο κρυπτοσύστημα δημοσίου κλειδιού που προτάθηκε είναι αυτό του McEliece[15] το 1978. Το σύστημα αυτό ακόμη παραμένει ασφαλές μέχρι σήμερα πάρα την ύπαρξη της θεώρησης περί κβαντικών υπολογιστών. Στην ουσία αυτό που αξιοποιείται στα κρυπτοσυστήματα που βασίζονται στη θεωρία κωδίκων είναι ότι για την κρυπτογράφηση εισάγουμε επί τούτου σφάλματα στο μήνυμα c πριν την αποστολή, τα οποία καλείται ο παραλήπτης να βρει και να διορθώσει. Το δημόσιο κλειδί του παραλήπτη είναι ο πίνακας γεννήτορας G του κώδικα αυτού. Ο αποστολέας κρυπτογραφεί το μήνυμα μετατρέποντάς το αρχικά σε μια λέξη του κώδικα και έπειτα προσθέτει λάθη από ένα μυστικό διάνυσμα σφάλματος e με βάρος t, δηλαδή $c = mG \oplus e$. Ο παραλήπτης αποκωδικοποιεί το c και λαμβάνει το m . Για να είναι αυτό το κρυπτοσύστημα ασφαλές πρέπει κάποιος επιτιθέμενος να μην μπορεί να αντιστοιχίσει τον εν λόγω «τυχαίο» κώδικα με κάποιον από τους γνωστούς κώδικες. Ο γεννήτορας G ο οποίος περιγράφει τον κώδικα, που είναι και το δημόσιο κλειδί, θα πρέπει να έχει τυχαία δομή έτσι ώστε να μην μπορεί κάποιος να καταλάβει τι ακριβώς κώδικας έχει χρησιμοποιηθεί – δηλαδή ο G να μην αντιστοιχεί σε κάποιον από τους γνωστούς κώδικες. Αν καταλάβει ο επιτιθέμενος τι κώδικας χρησιμοποιήθηκε (δηλαδή αν αναγνωρίσει ότι πρόκειται για κάποιον γνωστό κώδικα), άμεσα θα ξέρει και τι αλγόριθμο μπορεί να εφαρμόσει για να επιτεθεί στο συγκεκριμένο κρυπτοσύστημα και να αποκρυπτογραφήσει το C. Το κρυπτοσύστημα McEliece περιγράφει τον υπολογισμό του γεννήτορα G κατά τρόπο τέτοιο ώστε, αν και ο κώδικας είναι τυχαίος, ο παραλήπτης να μπορεί να υπολογίσει το αρχικό μήνυμα με το μυστικό κλειδί του. Ο αλγόριθμος αυτός χρησιμοποιεί δυαδικούς κώδικες Goppa και χρειάζεται αρκετά μεγάλα κλειδιά της τάξεως των kb. Μια άλλη πρόταση σχετική με τον αλγόριθμο McEliece είναι το κρυπτοσύστημα Niederreiter[19] που προσθέτει κάποιες βελτιώσεις στους αλγορίθμους κωδικοποίησης και αποκωδικοποίησης και μειώνει έτσι το υπολογιστικό κόστος αλλά και το μέγεθος κλειδιού. Το κρυπτοσύστημα Niederreiter χρησιμοποιεί λίγο διαφορετική προσέγγιση από το κρυπτοσύστημα McEliece στη δημιουργία δημοσίου κλειδιού. Η κοινότητα της μετακβαντικής κρυπτογραφίας έχει μεγάλες προσδοκίες για τα προαναφερθέντα δυο συστήματα δημοσίου κλειδιού. Το μόνο πρόβλημα σε αυτά τα συστήματα είναι το μεγάλο μέγεθος κλειδιού. Έχουν γίνει προσπάθειες συρρίκνωσης του δημοσίου κλειδιού με κάποιες τεχνικές συμπίεσης αλλά φαίνεται ότι τα κρυπτοσυστήματα με συμπιεσμένα κλειδιά είναι ευπαθέστερα σε κρυπταναλυτικές επιθέσεις. Η μετακβαντική κρυπτογραφία με χρήση θεωρίας κωδίκων προσφέρει καλές προοπτικές και σε σχήματα ψηφιακής υπογραφής και μονόδρομων συναρτήσεων.

5.5.2 Η κρυπτογραφία πλέγματος

Η κρυπτογραφία lattice βασίζεται στο πρόβλημα της μικρότερης διανυσματικής απόστασης σε ένα πλέγμα. Ως πλέγμα ορίζεται ένα σύνολο σημείων σε έναν χώρο ν διαστάσεων με μια περιοδική δομή. Είναι υπολογιστικά πολύ δύσκολο να βρεθεί το ελάχιστο διάνυσμα σε ένα πολυδιάστατο πλέγμα.

Η βασική ιδέα για τη δημιουργία ενός δημοσίου κλειδιού και γενικότερα ενός κρυπτοσυστήματος δημοσίου κλειδιού είναι η χρήση ενός καλά οργανωμένου δικτυώματος βάσης s πολλών διαστάσεων ως ιδιωτικό κλειδί και, ως δημόσιο κλειδί, μια εκδοχή p του δικτυώματος βάσης s που θα είναι διαφοροποιημένη. Έτσι για την κρυπτογράφηση ενός μηνύματος ο αποστολέας επιλεγεί ένα σημείο m του πολυδιάστατου χώρου p (του διαφοροποιημένου χώρου και όχι του αρχικού s) για το μήνυμα που θα αποστείλει. Έπειτα προσθέτει ένα τυχαίο λάθος στο επιλεγμένο σημείο m ώστε να προκύψει ένα νέο σημείο c που όμως βρίσκεται εγγύτερα στο αρχικό σημείο m σε σχέση με οποιοδήποτε άλλο σημείο του πλέγματος p. Αυτό το νέο σημείο c αποτελεί το κρυπτοκείμενο το οποίο ο αποστολέας στέλνει στον παραλήπτη. Αφού ο παραλήπτης γνωρίζει το ιδιωτικό κλειδί s μπορεί να αποκρυπτογραφήσει το σημείο c στο σημείο m που είναι και το αρχικό μήνυμα με σχετικά μικρό υπολογιστικό κόστος, λόγω του ότι τα πλέγματα p (δημόσιο κλειδί) και s (ιδιωτικό κλειδί) έχουν δημιουργηθεί με τρόπο τέτοιο ώστε γνώση και των δύο να επιτρέπει την εύρεση, για ένα σημείο του χώρου, του εγγύτερου με αυτό, σημείου στο πλέγμα. Το επίπεδο ασφάλειας χαρακτηρίζεται ισχυρό αφού ο επιτιθέμενος κατέχει μόνο το δημόσιο κλειδί p που είναι κάποιος μετασχηματισμός του s και θα χρειαστεί πολύ μεγάλη ισχύ και χρόνο για να αποκρυπτογραφήσει το c στο m. Το πρόβλημα της εύρεσης του σημείου ελάχιστης απόστασης σε ένα πλέγμα p είναι το γνωστό πρόβλημα closest vector problem(CVP) αλλά και η ανάκτηση του αρχικού χώρου βάσης s είναι το πρόβλημα εύρεσης της μικρότερης απόστασης -shortest vector problem (SVP): για τα δύο αυτά προβλήματα πιστεύεται ότι είναι υπολογιστικά αδύνατη η λύση τους, ακόμη και σε κβαντικό υπολογιστή. Το NTRUencrypt αποτελεί ένα ήδη εμπορικό προϊόν της μετακβαντικής κρυπτογραφίας πλέγματος. Η πατέντα αυτού ανήκει στην εταιρεία NTRU Cryptosystems.



Εικόνα 5.4: Κωδικοποίηση και αποκωδικοποίηση σε σύστημα πλέγματος[18].

5.5.3 Η μετακβαντική κρυπτογραφία συναρτήσεων κατακερματισμού

Η μετακβαντική κρυπτογραφία συναρτήσεων κατακερματισμού είναι εντελώς διαφορετική από τους δύο προαναφερθέντες κλάδους της θεωρίας κωδίκων και των πλεγμάτων. Οι συναρτήσεις κατακερματισμού (hash functions) είναι μονόδρομες (μιας κατεύθυνσης) συναρτήσεις, οι οποίες μπορούν να δεχτούν ως είσοδο οποιαδήποτε συμβολοσειρά (string) ανεξαρτήτως μεγέθους και παράγουν έξοδο μία συμβολοσειρά συγκεκριμένου μεγέθους, που αποκαλείται αποτύπωμα (digest) ή κατακερματισμένη τιμή (hash value) της εισόδου. Υπάρχουν τρεις ιδιότητες μιας κρυπτογραφικής συνάρτησης κατακερματισμού.

i. Preimage resistance

Πρέπει να είναι υπολογιστικά δύσκολο να υπολογιστεί η είσοδος από δοθείσα έξοδο, δηλαδή να υπολογιστεί είσοδος που να παράγει συγκεκριμένο αποτύπωμα. Για αυτό και άλλωστε θεωρούνται συναρτήσεις μιας κατεύθυνσης ή μονόδρομες (one-way).

ii. Second preimage resistance:

Δοθείσης μιας συμβολοσειράς και του αποτυπώματος αυτής, πρέπει να είναι υπολογιστικά δύσκολο να βρεθεί μια άλλη συμβολοσειρά με το ίδιο αποτύπωμα με αυτό της αρχικής.

iii. Collision resistance

Πρέπει να είναι δύσκολο να βρεθούν δυο οποιεσδήποτε διαφορετικές συμβολοσειρές που να έχουν το ίδιο αποτύπωμα.

Ο αλγόριθμος Grover[11] βελτιώνει το χρόνο εξαντλητικής αναζήτησης (brute force) κατά παράγοντα τετραγωνικής ρίζας, αναφορικά με την ιδιότητα «preimage resistance». Οι πιο γνωστοί κλασσικοί αλγόριθμοι εύρεσης «συγκρούσεων» (collisions) σε συναρτήσεις κατακερματισμού βασίζονται στο λεγόμενο παράδοξο των γενεθλίων και δίδουν πάλι μια μείωση υπολογιστικού χρόνου της τάξης της τετραγωνικής ρίζας σε σχέση με τη μέθοδο brute force. Παρόλα αυτά η επίπτωση της μείωσης του χρόνου για εύρεση συγκρούσεων με τον αλγόριθμο Grover δεν είναι αποδεκτή ευρέως από την επιστημονική κοινότητα. Ένα σπουδαίο χαρακτηριστικό των μονόδρομων συναρτήσεων είναι το γεγονός ότι δεν επηρεάζονται από τον αλγόριθμο Shor πολυωνυμικού χρόνου. Για αυτό το λόγο είναι υποψήφιες συναρτήσεις για μετακβαντικά σχήματα δημοσίου κλειδιού. Εντούτοις, δεν είναι ακόμη γνωστό πώς μπορεί κάνει να κατασκευάσει κάποιος κρυπτοσύστημα δημοσίου κλειδιού το οποίο να βασίζεται σε συναρτήσεις hash αφού εξ ορισμού είναι δύσκολο να υπολογιστεί το αντίστροφο μίας τέτοιας συνάρτησης. Αντιθέτως είναι πιο εύκολο να κατασκευαστούν σχήματα ψηφιακών υπογράφων που θα χρησιμοποιούν τέτοιες συναρτήσεις σαν δομικά στοιχεία. Παραδείγματος χάριν θεωρούμε τα παρακάτω:

Ο Α θέλει να υπογράψει ένα μήνυμα που αποτελείται από μόνο ένα bit. Δημιουργεί ένα ιδιωτικό κλειδί υπογραφής με την τυχαία επιλογή δυο συμβολοσειρών r0, r1. Υπολογίζει το δημόσιο κλειδί ως το ζεύγος {s0 = h(r0), s1 = h(r1)} και κοινοποιεί τις τιμές αυτές (s0 , s1).

Ο Β παραλαμβάνει το ζεύγος (s0, s1) και ξέρει ότι ανήκει στον Α.

Έστω ότι οι δύο πιθανές τιμές του μηνύματος είναι 1 (αληθές – true) και 0 (ψευδές – false).

Για να υπογράψει ο Α το μήνυμα true (1) κοινοποιεί το r1.

Ο Β μπορεί να υπολογίσει το h(r1) και να το συγκρίνει με το δημόσιο κλειδί s1: άρα, να πιστοποιήσει την προέλευση του μηνύματος βάσει δημοσίου κλειδιού.

Η υπογραφή προφανώς είναι του Α αφού μόνο εκείνος γνώριζε το preimage r1 του s1 και βάσει της ιδιότητας του Preimage resistance μόνο εκείνος μπορεί να είναι ο αποστολέας, αφού είναι υπολογιστικά αδύνατο να βρεθεί από κάποιον άλλο το preimage του r1.

Βέβαια εφεξής δε δύναται ο Α να ξαναχρησιμοποιήσει την ιδιά υπογραφή διότι αν κοινοποιήσει και το r0 διαμοιράζεται, τελικά, το ιδιωτικό του κλειδί στο σύνολό του. Αυτό είναι μειονέκτημα στο σχήμα που παρουσιάζεται. Άλλο ένα μειονέκτημα προφανώς έχει να κάνει με το μήκος του μηνύματος προς υπογραφή.

5.5.4 Η κρυπτογραφία πολλών μεταβλητών

Η κρυπτογραφία πολλών μεταβλητών βασίζεται στη δυσκολία επίλυσης συστημάτων εξισώσεων βαθμού δύο (δηλ. τετραγωνικών) με πολλές μεταβλητές (multivariate quadratic) σε πεπερασμένα σώματα: είναι πρόβλημα με πολυπλοκότητα NP-hard. Αντίθετα με τα γραμμικά συστήματα, για τα οποία υπάρχουν τεχνικές επίλυσης, δεν υπάρχει αποδοτικός αλγόριθμος που να επιλύει τυχαία πολυωνυμικά συστήματα πολλών μεταβλητών. Η δυσκολία επίλυσης έγκειται στο μέγεθος του υποκείμενου διανυσματικού πεδίου, στον αριθμό μεταβλητών και τον βαθμό του συστήματος. Αν υποθέσουμε ότι ο αριθμός εξισώσεων αλλά και των μεταβλητών είναι αρκούντως μεγάλο, ακόμα και στο μικρότερο πεπερασμένο σώμα GF(2) – δηλαδή το σώμα που αποτελείται μόνο από τα στοιχεία 0 και 1 - είναι αρκετά δύσκολο να επιλυθούν. Τα πολυωνυμικά συστήματα πολλών μεταβλητών μπορούν θεωρητικά να επιλυθούν με διάφορους αλγόριθμους, οι οποίοι όμως δεν είναι αποδοτικοί. Σε τεχνικές εξαντλητικής αναζήτησης αυτό που γίνεται είναι ότι δοκιμάζονται όλες οι πιθανές τιμές μεταβλητών μέχρι την εύρεση σωστής λύσης. Αυτό βέβαια χρειάζεται εκθετικά μεγάλο χρόνο σε σχέση με το πλήθος των μεταβλητών. Μια πιο αποδοτική προσέγγιση είναι η αριθμητική επίλυση του συστήματος. Υπάρχουν αλγόριθμοι με διάφορες προσεγγίσεις με πιο γνώστη την οικογένεια αλγορίθμων στα πεπερασμένα σώματα GF(4) και GF(5) . Αν υποθέσουμε την ύπαρξη του κβαντικού υπολογιστή, ο αλγόριθμος Grover δίνει επιτάχυνση ως προς το χρόνο της τάξης της τετραγωνικής ρίζας για κρυπτανάλυση εξαντλητικής αναζήτησης (brute force). Για την δημιουργία ενός συστήματος δημοσίου κλειδιού με βάση τις πολυωνυμικές συναρτήσεις πολλών μεταβλητών, το δημόσιο κλειδί είναι ένα σύνολο από τετραγωνικά πολυώνυμα πολλών μεταβλητών και το ιδιωτικό κλειδί είναι μια «κερκόπορτα» που επιτρέπει την αποδοτική και άμεση επίλυση του συστήματος. Τις περισσότερες φορές η κερκόπορτα κατασκευάζεται υπολογίζοντας το δημόσιο κλειδί P από m πολυώνυμα με n μεταβλητές ως γινόμενο τριών επάλληλων πινάκων T, S και ενός κυβικού αντιστρέψιμου Q.Παράδειγμα:

P = T*Q*S

Για να υπογράψει ο Α ένα μήνυμα z υπολογίζει την υπογραφή w ως εξής:

Z'= hash(z)

$$y = T^{-1}(z'), x = Q^{-1}(y), w = S^{-1}(x).$$

Ο Β μπορεί απλά να πιστοποιήσει την εγκυρότητα της υπογραφής με το δημόσιο κλειδί P ελέγχοντας ότι hash(z) = P(w). Εδώ σημειώνεται ότι το δημόσιο κλειδί P πρέπει να κατασκευάζεται κατά τρόπον ώστε το σύστημα να μην μπορεί να αναστραφεί από επίδοξο εισβολέα.

$$\begin{aligned} x_0x_3 + x_2x_3 + x_0 + 1 &= 0 \\ x_0x_1 + x_2x_3 + x_2 + 1 &= 0 \\ x_0x_1 + x_0x_3 + x_0 + x_1 + 1 &= 0 \\ x_1x_2 + x_2x_3 + x_3 &= 0 \end{aligned}$$

Εικόνα 5.5: Σύστημα 4 εξισώσεων με 4 μεταβλητές στον σώμαGF(2) ενώ μια προφανής λύση είναι η (0,1,0) [18]

5.6 Σύγκριση σχημάτων μετακβαντικής κρυπτογραφίας

Τα προαναφερθέντα συστήματα μετακβαντικής κρυπτογραφίας ποικίλουν πολύ σε σχέση με την υπολογιστική ισχύ και τους πόρους που χρειάζονται. Στο παρακάτω σχήμα φαίνεται πολύ καλά η σύγκριση διάφορων μεγεθών στα υπό συζήτηση μετακβαντικά συστήματα .Εν γένει θα μπορούσαμε να παρατηρήσουμε ότι όλα τα κρυπτοσυστήματα που είναι υποψήφια για υλοποίηση έχουν μεγαλύτερα μεγέθη κλειδιών, κείμενων, κρυπτοκειμένων και υπογραφών, σε σχέση με τα κλασσικά συστήματα. Παρόλα αυτά, αυτό το κόστος ως προς τα μεγέθη είναι το τίμημα για συστήματα ανθεκτικά στην επίθεση με κβαντικό υπολογιστή, ενώ αποκλείονται κλασσικά σημερινά συστήματα όπως τα RSA, και κρυπτοσυστήματα ελλειπτικών καμπυλών (ECC). Στο μεγαλύτερο μέρος των ερευνών ως προς την ελαχιστοποίηση των μεγεθών όπως τα αναφέραμε, προκύπτουν μεν καλυτερεύσεις αλλά από την άλλη δημιουργούνται συστήματα πιο

ευπαθή. Μάλιστα, για μερικά τέτοια κρυπτοσυστήματα δεν είναι δυνατή η ασφαλής μείωση των δεδομένων η του μεγέθους κλειδιών. Πολύ καλά θεωρούνται τα συστήματα ψηφιακών υπογράφων καθώς και τα κρυπτοσυστήματα δημοσίου κλειδιού McEliece - Niederreiter και επικρατούν, προς το παρόν, στην μετακβαντική κρυπτογραφία δημοσίου κλειδιού σε σχέση με τις υπόλοιπες προτάσεις.

Scheme		**Public key size** (bytes)	**Data size** (bytes)
Public-key signatures:			
• Hash based:			
– XMSS (stateful)	[17]	64	2,500 – 2,820
– SPHINCS (state free)	[9]	1,056	41,000
• Multivariate based:			
– HFEv-*	[51]	500,000 – 1,000,000	25 – 32
Public-key encryption:			
• Code based:			
- McEliece	[10]	958,482 – 1,046,739	187 – 194
• Lattice based:			
– NTRUEncrypt	[35, 37]	1,495 – 2,062	1,495 – 2,062
Key exchange:			
• Lattice based:			
– NewHope	[3]	—	1,824 – 2,048
• Supersingular isogenies:			
– SIDH	[21]	—	564
Classical schemes:			
• RSA:			
– RSA-2048		256	256
– RSA-4096		512	512
• ECC:			
– 256-bit		32	32
– 512-bit		64	64
• Key exchange:			
– DH		—	256 – 512
– ECDH		—	32 – 64

* Values using field $\mathbb{F}_2$ and parameter n (number of variables) between 200 and 256.

Εικόνα 5.6: Σύγκριση μεγεθών υφιστάμενων παραμέτρων διάφορων κρυπτοσυστημάτων [18].

Κεφάλαιο 6
Θεωρία κωδίκων - Κώδικες Goppa

6.1 Βασικά στοιχεία κωδίκων

Σε αυτό το κεφάλαιο θα κάνουμε μια σύντομη αναφορά στις βασικές έννοιες από τη θεωρία κωδίκων ώστε να γίνει κατανοητή η δομή των κωδίκων Goppa που θα χρειαστούν για την παρουσίαση του συστήματος McEliece σε επόμενο κεφάλαιο.

Σύμφωνα με τη θεωρία κωδίκων, ο κώδικας είναι στην ουσία μια γλώσσα που έχουμε ορίσει σαφώς. Για παράδειγμα, ακόμη και η ελληνική γλώσσα ή οποιαδήποτε άλλη, σαφώς ορισμένη γλώσσα επικοινωνίας, είναι ένας κώδικας. Σε μια γλώσσα τα μηνύματα μεταδίδονται γραπτά η προφορικά αλλά είναι σαφώς ορισμένα. Χρησιμοποιώντας τα 24 γράμματα της ελληνικής γλώσσας μπορούμε να φτιάξουμε θεωρητικά $\binom{24}{1}+\binom{24}{2}+\ldots+\binom{24}{\nu}$ λέξεις. Στην πραγματικότητα αυτό δεν γίνεται, διότι υπάρχουν κανόνες που ορίζουν τις λέξεις και την ορθότητά τους σε μια γλώσσα άρα και σε έναν κώδικα. Ένα άλλο γνώρισμα και καθημερινή εφαρμογή ενός κώδικα είναι η δυνατότητα καλής κατανόησης ενός μηνύματος. Για παράδειγμα ο

Α στέλνει ένα μήνυμα στο Β μέσα από ένα θορυβώδες κανάλι επικοινωνίας. Ο Β λαμβάνει ένα μήνυμα με κάποια στοιχεία αλλοιωμένα από το θόρυβο. Μπορεί ο Β να καταλάβει ποια λέξη έχει παραμορφωθεί από τό θόρυβο και να την αντιστοιχίσει σε γνωστή λέξη της γλώσσας (κώδικα)που χρησιμοποιείται και αυτό αποτιμά την ποιότητα ενός κώδικα και είναι η δυνατότητα διόρθωσης λαθών (διορθωτική ικανότητα).

Αλφάβητο ενός κώδικα είναι το $\mathbb{A} = \{ a1 , a2 , \dots , a_r \}$ που είναι ένα τυχαίο πεπερασμένο σύνολο. και αποτελείται από στοιχεία που λέγονται γράμματα ή χαρακτήρες. Μια ακολουθία στοιχείων του $\mathbb{A}$ θα ονομάζεται λέξη και συνήθως συμβολίζεται με ένα γράμμα λατινικού αλφάβητου σε έντονη (bold) γραφή.

Με $\mathbb{A}^n$ θα συμβολίζουμε το σύνολο όλων των λέξεων με χαρακτήρες από το αλφάβητο $\mathbb{A}$, οι οποίες έχουν μήκος n.

Έστω $x = a_1 a_2 \dots a_n \; και \; y = b_1 b_2 \dots b_n \in \mathbb{A}^n$.

Η απεικόνιση $d : \mathbb{A}^n \times \mathbb{A}^n \longrightarrow \mathbb{Z}$ που φαίνεται παρακάτω ως εξής:

$$d(x,y) = \sum_{i=1}^{n} r_i, \mu\varepsilon \; r_i = \begin{cases} 0, a_i = b_i \\ 1, a_i \neq b_i \end{cases}$$

ορίζεται ως απόσταση Hamming (Hamming distance) των κωδικολέξεων x,y και μας δείχνει τα ψηφία στα οποία διαφέρουν οι κωδικολέξεις x, y.

Έστω ένα αλφάβητο $\mathbb{A}$. Κάθε μη μηδενικό υποσύνολο $\mathcal{C}$ του $\mathbb{A}*$ ονομάζεται κώδικας επί του αλφάβητου $\mathbb{A}$ και τα στοιχεία του κωδικολέξεις (ή κωδικές λέξεις).

Έστω $S = \{a_1 a_2 \dots a_s\}$ ένα σύνολο το οποίο ονομάζεται πηγή ή μήνυμα και $\mathcal{C}$ κάποιος κώδικας. Ορίζεται ως συνάρτηση κωδικοποίησης κάθε συνάρτηση $f : S \longrightarrow \mathcal{C}$.

Η διαδικασία κωδικοποίησης με γνωστά τα σύνολα S και $\mathcal{C}$, εξαρτάται από την επιλογή και εφαρμογή της συνάρτησης κωδικοποίησης f.

A → 1000001	$J \rightarrow 1001010$	$S \rightarrow 1010011$
$B \rightarrow 1000010$	$K \rightarrow 1001011$	$T \rightarrow 1010100$
$C \rightarrow 1000011$	$L \rightarrow 1001100$	$U \rightarrow 1010101$
$D \rightarrow 1000100$	$M \rightarrow 1001101$	$V \rightarrow 1010110$
$E \rightarrow 1000101$	$N \rightarrow 1001110$	$W \rightarrow 1010111$
$F \rightarrow 1000110$	$O \rightarrow 1001111$	$X \rightarrow 1011000$
$G \rightarrow 1000111$	$P \rightarrow 1010000$	$Y \rightarrow 1011001$
$H \rightarrow 1001000$	$Q \rightarrow 1010001$	$Z \rightarrow 1011010$
$I \rightarrow 1001001$	$R \rightarrow 1010010$	$Space \rightarrow 0100000$

Εικόνα 6.1: Κωδικοποίηση του λατινικού αλφάβητου στο δυαδικό σύστημα[31].

Ένας κώδικας $\mathcal{C}$ είναι κώδικας σταθερού μήκους όταν και μόνο όταν οι κωδικολέξεις που περιέχονται σε αυτόν έχουν όλες ίδιο μήκος, δηλαδή υπάρχει θετικός n, με $\mathcal{C} \subseteq \mathbb{A}^n$. Ο αριθμός n είναι το μήκος του κώδικα. Διαφορετικά, ο κώδικας λέγεται μεταβλητού μήκους. Συνήθως χρησιμοποιούμε κώδικες σταθερού μήκους γιατί είναι ευκολά κατανοητό ότι με κώδικες μεταβλητού μήκους είναι αρκετά δύσκολος ο εντοπισμός του τέλους μιας λέξης και της αρχής της επομένης [31].

Στη συνέχεια, θα ασχοληθούμε με κώδικες σταθερού μήκους επί πεπερασμένου σώματος $\mathbb{F}$. Επομένως, ένας κώδικας για την χρήση του στην παρούσα εργασία χαρακτηρίζεται, από τις εξής παραμέτρους:

i. το αλφάβητο που είναι το σώμα $\mathbb{F}$,

ii. το μήκος n

iii. το μέγεθός του $M = |\mathcal{C}|$.

Ενώ χάριν συντομίας θα αναφέρεται ως ένας (n, M) κώδικας.

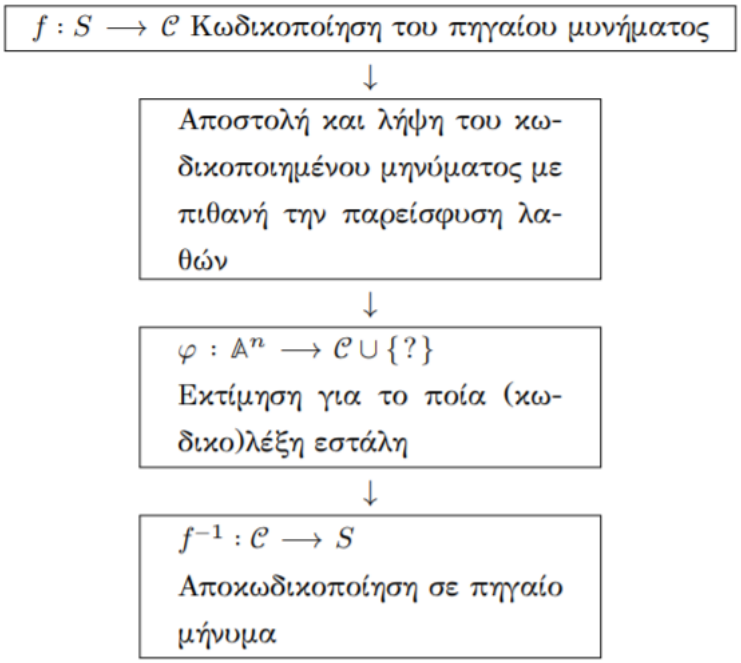
Ένα πλεονέκτημα της χρήσης ενός κώδικα σταθερού μήκους επί του πεπερασμένου σώματος $\mathbb{F}$ όπως αυτό ορίζεται κάθε φορά, είναι ότι το σύνολο $\mathbb{F}^n$ των λέξεων μήκους n είναι στην πραγματικότητα διανυσματικός χώρος και ως τέτοιον τον μελετούμε. Άρα επιτρέπεται να χρησιμοποιούμε τις ιδιότητες διανυσματικού χώρου στη μελέτη των κωδίκων.

Έστω $\mathbf{a} \in \mathbb{F}^n$. Η απόσταση της **a** από τη μηδενική λέξη **0** = (0, 0, … , 0) ορίζεται ως βάρος της λέξεως **a** και συμβολίζεται με $w(\mathbf{a})$. Δηλαδή $w(\mathbf{a}) = d(\mathbf{a}, \mathbf{0})$. Συνεπώς, το βάρος μιας λέξης μας δίνει τον αριθμό των μη μηδενικών στοιχείων της.

Κατά την κωδικοποίηση μηνύματος, όπως έχουμε πει, εφαρμόζεται η συνάρτηση κωδικοποίησης f του μηνύματος. Έπειτα, το κωδικοποιημένο μήνυμα (κωδική λέξη) αποστέλλεται μέσα από ένα, ενδεχομένως θορυβώδες, κανάλι - για παράδειγμα, μέσω μιας σύνδεσης τύπου socket σε επίπεδο δικτύου ή και σε φυσικό επίπεδο (π.χ. αλλοιώσεις ηλεκτρομαγνητικών κυμάτων). Επίσης το ίδιο μήνυμα μπορούμε να το αποθηκεύουμε σε ψηφιακά μέσα αποθήκευσης, όπως για παράδειγμα σε ένα σκληρό δίσκο. Για την αποκωδικοποίηση ακολουθείται η αντίστροφη διαδικασία. άρα, χρησιμοποιείται η αντίστροφη f^{-1} της συνάρτησης κωδικοποίησης f . Σε ιδεατά συστήματα τηλεπικοινωνίας δε υπάρχει κάποιο εμπόδιο, αρκεί ο Α να γνωρίζει την f και ο παραλήπτης Β την συνάρτηση αποκωδικοποίησης που είναι η αντίστροφή της, δηλαδή η f^{-1} . Στην πραγματικότητα όμως έχουμε θόρυβο στο κανάλι επικοινωνίας οπότε στην καλύτερη περίπτωση επέρχεται αλλοίωση στις κωδικολέξεις που μεταδίδονται. Συνεπώς η αντίστροφη συνάρτηση δεν μπορεί να προσφέρει πολλά στη διόρθωση σφαλμάτων λόγω θορύβου η αστοχίας υλικού. Έτσι, η θεωρία κωδίκων δύναται να μας βοηθήσει σημαντικά στην εύρεση και αντιμετώπιση λαθών που έχουν υπεισέλθει [31].

Έστω ότι στέλνουμε ένα μήνυμα με χρήση του κώδικα $\mathcal{C} \subseteq \mathbb{F}^n$ (συζητάμε πάντα επί ενός πεπερασμένου σώματος $\mathbb{F}$). Όταν στέλνεται μία κωδικολέξη, **a**, ενδέχεται, όπως έχει επισημανθεί, ο παραλήπτης να λάβει μια άλλη λέξη **b**. Η διαφορά $\mathbf{e} = \mathbf{b} - \mathbf{a} \in \mathbb{F}^n$ λέγεται διάνυσμα λάθους ή απλώς λάθος το οποίο εισήχθη κατά τη μετάδοση του μηνύματος.

Έστω $\mathcal{C}$ κώδικας μήκους n με χαρακτήρες από το αλφάβητο $\mathbb{A}$. Ορίζουμε ως κανόνα αποφασησιμότητας τη συνάρτηση $\varphi : \mathbb{A}^n \longrightarrow \mathcal{C}$, βάσει της οποίας αποδεχόμαστε πως ελήφθη η λέξη $\mathbf{x} \in \mathbb{A}^n$, και ότι εστάλη η λέξη $\varphi(\mathbf{x})=\boldsymbol{\alpha}$, αν $\varphi(\mathbf{x}) \in \mathcal{C}$ ή ότι πρόκειται για λάθος στη μετάδοση αν $\varphi(\mathbf{x}) \notin \mathcal{C}$. Εδώ πρέπει να τονισθεί ότι η συνάρτηση φ, που αναφέρεται στον ορισμό, είναι το στάδιο που προηγείται της εφαρμογής της αντίστροφης συνάρτησης f^{-1}, όπου f είναι η συνάρτηση κωδικοποίησης κατά τα γνωστά. Σχηματικά φαίνεται στην παρακάτω εικόνα η όλη διαδικασία που περιγράψαμε πιο πάνω.



Εικόνα 6.2:Κωδικοποίηση και αποκωδικοποίηση μηνύματος [31]

Για να γίνει κατανοητή η αποκωδικοποίηση δίνουμε ένα απλό παράδειγμα. Έστω κώδικας $\mathcal{C}$ = { 000, 111,110} και ο παραλήπτης λαμβάνει την κωδικολέξη 001, η οποία είναι προφανές πως δεν ανήκει στον κώδικα που ορίσαμε. Μπορούμε εύκολα να υποθέσουμε ότι το πιθανότερο σενάριο είναι να έχει αποσταλεί η λέξη 000 διότι είναι η πλησιέστερη, σε σχέση με τη λέξη που ελήφθη, λέξη που ανήκει στον κώδικά μας. Οπότε αποκωδικοποιείται ως αποσταλείσα κωδικολέξη η 000.

Φαίνεται ξεκάθαρα ότι αν έχει σταλεί κωδικολέξη $\boldsymbol{\alpha} \in \mathcal{C}$ και έχει παραληφθεί η λέξη $\mathbf{x}$, η αποκωδικοποίηση κρίνεται ότι έχει γίνει σωστά αν $\varphi(\mathbf{x}) = \alpha$.

Ορίζουμε συνάρτηση αποκωδικοποίησης ως προς την πλησιέστερη κωδικολέξη ως εξής:

Έστω ότι έχει σταλεί μια λέξη $\mathbf{c} \in \mathcal{C}$ και έχει ληφθεί η λέξη $\mathbf{x} \in \mathbb{A}$. Έχουμε τότε δυο υποπεριπτώσεις:

Αν η $\mathbf{x}$ ανήκει στον κώδικα, τότε $\varphi(\mathbf{x}) = \mathbf{x}$, δηλαδή δεχόμαστε ότι εστάλη η λέξη $\mathbf{x}$.

Αν η $\mathbf{x}$ δεν ανήκει στον κώδικά μας , πρέπει να υπολογίσουμε την απόσταση της $\mathbf{x}$ από όλες τις λέξεις του κώδικα. Υπάρχει λέξη $\mathbf{d}$ του κώδικα με την μικρότερη απόσταση από την $\mathbf{x}$. Αν η λέξη $\mathbf{d}$ είναι η μοναδική με την ιδιότητα αυτή, τότε ορίζουμε $\varphi(\mathbf{x}) = \mathbf{d}$ και κατά την αποκωδικοποίηση

δεχόμαστε την **d**. Αν υπάρχουν περισσότερες λέξεις οι οποίες απέχουν την ίδια ελάχιστη απόσταση από τη **x**, τότε αδυνατούμε να αποφανθούμε για το αποτέλεσμα και ορίζουμε $\varphi(x) = ?$. Η αποκωδικοποίηση αυτού του τύπου με τη συνάρτηση φ λέγεται αποκωδικοποίηση ως προς την αρχή της πλησιέστερης λέξης.

Έστω $\mathcal{C}$ κώδικας που περιέχει αρκετά στοιχεία. Η ελάχιστη απόσταση $d(\mathcal{C})$ του $\mathcal{C}$ είναι η ελάχιστη δυνατή απόσταση μεταξύ δύο κωδικών λέξεων.

Δηλαδή: $d(\mathcal{C}) = \min\{ d(\mathbf{c}, \mathbf{d}) \mid \mathbf{c}, \mathbf{d} \in \mathcal{C}, \mathbf{c} \neq \mathbf{d} \}$.

Άρα, εκτός από το αλφάβητο, το μήκος n και το μέγεθος $M = |\mathcal{C}|$ ενός κώδικα $\mathcal{C}$, χρησιμοποιώντας την αρχή της πλησιέστερης λέξης εισάγεται μια επιπλέον παράμετρος, που είναι η ελάχιστη απόσταση $d = d(\mathcal{C})$ του κώδικα, και πλέον θα ορίζουμε ένα κώδικα ως (n, M, d) κώδικα.

Ορίζουμε διάνυσμα λάθους $\mathbf{e} \in \mathbb{A}_n$ το οποίο δύναται να ανιχνεύεται από κώδικα $\mathcal{C} \subseteq \mathbb{A}_n$, αν η λέξη $\mathbf{a}+\mathbf{e}$ δεν ανήκει στο κώδικα $\mathcal{C}$ για κάθε κωδικολέξη $\mathbf{a} \in \mathcal{C}$. Αν όμως βρούμε $\mathbf{a} \in \mathcal{C}$ τέτοιο ώστε $\mathbf{a} + \mathbf{e} \in \mathcal{C}$, τότε το διάνυσμα λάθους **e** είναι μη ανιχνεύσιμο.

Γενικά ένας κώδικας $\mathcal{C}$ ανιχνεύει λ το πλήθος λάθη, όπου $\lambda > 0$ και ακέραιος αριθμός, αν και μόνο αν κάθε διάνυσμα **e** και βάρους $\leq \lambda$ ανιχνεύεται από τον κώδικα $\mathcal{C}$.

Έστω κώδικας $\mathcal{C}$ με ελάχιστη απόσταση $d(\mathcal{C}) = d$. Σύμφωνα με τα προηγούμενα, θα ανιχνεύει (μέχρι) $d - 1$ το πλήθος λάθη. Αν χρησιμοποιηθεί για διόρθωση λαθών, θα διορθώνει $\left\lfloor \frac{d-1}{2} \right\rfloor$ λάθη. Συνήθως όμως ο κώδικας χρησιμοποιείται και για διόρθωση αλλά και για ανίχνευση λαθών. Η ταυτόχρονη χρήση κώδικα για διόρθωση και ανίχνευση λαθών πλεονεκτεί ως προς την εξοικονόμηση πόρων, χρόνου, χρημάτων. Φυσικά χρειάζεται προσοχή ως προς τη γνώση του μέγιστου αριθμού λαθών που μπορεί να ανιχνεύσει.

6.2 Προβλήματα NP-complete

Υπάρχει μια κλάση από προβλήματα που είναι εξαιρετικά δύσκολα και πιστεύεται δημόσια ότι δεν δύναται ένας κλασσικός υπολογιστής να τα επιλύσει. Ως πρόβλημα τύπου NP ορίζουμε ένα πρόβλημα που δεδομένης της λύσης του μπορεί εύκολα να ελεγχθεί η ορθότητά της. Τα προβλήματα αυτά επιλύονται σε πολυωνυμικό χρόνο από μη ντετερμινιστική μηχανή Turing (ενώ παραμένει ως κλασικό πρόβλημα της πληροφορικής ως προς το αν μπορούν να επιλυθούν σε πολυωνυμικό χρόνο από ντετερμινιστική μηχανή). Ένα πρόβλημα ορίζεται ως NP-complete αν είναι ένα από τα δυσκολότερα προβλήματα τύπου NP. Μέχρι τώρα δεν είμαστε σίγουροι πως ο κβαντικός υπολογιστής μπορεί να επιλύσει ένα πρόβλημα τέτοιου τύπου και θέλουμε να είμαστε αισιόδοξοι πως και στην πράξη δε θα μπορέσει. Άρα τέτοια προβλήματα είναι πολύ σημαντικά για την μετακβαντική κρυπτογραφία.

6.3 Το πρόβλημα αποκωδικοποίησης γραμμικού κώδικα

Το πρόβλημα αποκωδικοποίησης τυχαίου γραμμικού κώδικα, που ορίζεται ως η διαδικασία εύρεσης της πλησιέστερης κωδικολέξης, είναι ένα πρόβλημα αυτής της κατηγορίας - δηλαδή NP-complete. Δεν υπάρχει αλγόριθμος που να μπορεί να το επιλύσει σε πολυωνυμικό χρόνο αλλά μάλλον σε εκθετικό. Αν παρεμπιπτόντως κάποιος έβρισκε αλγόριθμο για την επίλυσή του, τότε θα κατέρριπτε και πολλά άλλα προβλήματα NP-complete.

Οι κώδικες διόρθωσης λαθών που σχεδιάζονται με συγκεκριμένο τρόπο – δηλαδή δεν είναι τυχαίοι - έχουν την ιδιότητα ότι η συνάρτηση – αλγόριθμος αποκωδικοποίησης μπορεί να διορθώσει, όπως είδαμε, ένα μέγιστο αριθμό λαθών r(ή t). Τα λάθη συνέβησαν είτε κατά τη διάρκεια αποστολής ενός μηνύματος είτε κατά την κωδικοποίησή του. Έτσι αν ο παραλήπτης λάβει το μήνυμα $y = c + e$ οπού c είναι η κωδικολέξη και e είναι το διάνυσμα που περιγράφει τις θέσεις και τον αριθμό των λαθών, δηλαδή είναι διάνυσμα με βάρος $\leq r$ μπορούμε με ευκολία να βρούμε την απεσταλμένη κωδικολέξη από το c. Άρα εκτός από την επίλυση του προβλήματος θορύβου σε κανάλι επικοινωνίας η σε ένα δίσκο, αυτή η τεχνική θα μπορούσε πολύ εύκολα να εφαρμοστεί σε κρυπτοσυστημα δημοσίου κλειδιού. Τούτο προκύπτει διότι ο επίδοξος εισβολέας δεν ξέρει τον τρόπο να διαχωρίσει τον θόρυβο e από τη λέξη c ενώ ο παραλήπτης έχει τις πληροφορίες που χρειάζεται για να το πράξει.

Έτσι ο κώδικας μπορεί απευθείας να εφαρμοστεί στο κρυπτοσυστημα δημοσίου κλειδιού όπου όλοι έχουν τη συνάρτηση κρυπτογράφησης μηνύματος, άλλα μόνο ο παραλήπτης έχει τη συνάρτηση αποκωδικοποίησης και άρα μόνο αυτός μπορεί να αποκωδικοποιήσει σταλθέν μήνυμα. Προφανώς σε αυτή την περίπτωση, δημόσιο κλειδί μπορεί να θεωρηθεί ο πίνακας κωδικοποίησης και ιδιωτικό κλειδί η συνάρτηση αποκωδικοποίησης. Βέβαια για να είναι σίγουρο ότι κανείς δεν μπορεί να εξάγει συμπεράσματα για την συνάρτηση αποκωδικοποίησης, δοθέντος του πίνακα κωδικοποίησης, πρέπει ο τελευταίος να έχει υποστεί κάπου είδους επεξεργασία ώστε να φαίνεται ως ένας ψευδοτυχαίος πίνακας. Επιπλέον ο κώδικας που θα χρησιμοποιείται πρέπει να ανήκει σε μεγάλη κλάση κωδίκων ώστε να μην είναι διακριτές οι επιλογές του κώδικα κάθε φορά για ευνόητους λόγους.

6.4 Οι κώδικες Goppa

Οι κώδικες Goppa αποτελούν μια μεγάλη κλάση αλγεβρικών κωδίκων διόρθωσης λαθών με αυτά τα χαρακτηριστικά που προαναφέραμε. Αποτελούν γενίκευση των κωδίκων BCH[31] οι οποίοι είναι μια μικρότερη κλάση κωδίκων σε σύγκριση με τους Goppa.

Ένας κώδικας Goppa Γ(L, g(z)) ορίζεται μονοσήμαντα από το πολυώνυμο Goppa g(z) βαθμού σε ένα πεπερασμένο σώμα GF(q^m) , όπου q πρώτος αριθμός, καθώς και από ένα υποσύνολο L του προαναφερθέντος πεπερασμένου σώματος.

$$g(z) = g_0 g_1 z + \cdots + g_t z^t = \sum_{i=0}^{t} g_i z^i$$

$$L = \{a_1, \ldots, a_n\} \subseteq GF\ (q^m)$$

Όπου g(a) μη μηδενικό για όλα τα a ∈ L

Σε κάθε διάνυσμα c=(c_1 c_2 … c_n) με τιμές στο GF(q) αντιστοιχούμε τη συνάρτηση Rc(z) ως εξής:

$$R_c(z) = \sum_{i=1}^{n} \frac{c_i}{z - a_i}$$

Όπου το πολυώνυμο $1/(z - a_i)$ ορίζεται ως το αντίστροφο του (z-α_i) (mod g(z)), δηλαδή

$$(z - a_i)\frac{1}{z-a_i} \equiv 1\ (mod\ g(z))$$

Ο κώδικας Goppa Γ(L, g(z)) αποτελείται από όλα τα διανύσματα c για τα οποία ισχύει $R_c(z) \equiv 0 (mod\, g(z))$ (δηλαδή το υπόλοιπο της διαίρεσής τους με το g(z) είναι το 0).

Οι παράμετροι του κώδικα Goppa, ο οποίος είναι γραμμικός κώδικας, είναι το μέγεθος n, η διάσταση k και η ελάχιστη απόσταση d. Χρησιμοποιείται συχνά ο ορισμός ενός Goppa ως [n,k,d] σύμφωνα με τις παραμέτρους που αναφέρθηκαν.

i. Η παράμετρος n αναφέρεται στο σταθερό μήκος των κωδικολέξεων

ii. Η διάσταση k ικανοποιεί την ανίσωση k≥n-mt

iii. Η ελάχιστη απόσταση d ≥t+1

Μας ενδιαφέρει ιδιαίτερα η περίπτωση του δυαδικού κώδικα Goppa όπου η ελάχιστη απόσταση d του κώδικα είναι η μεγαλύτερη δυνατή. Αυτή η περίπτωση είναι ο Goppa Γ(L, g(z)) με g(z) βαθμού t στο πεπερασμένο σώμα GF(2^m). Σε αυτή την περίπτωση η ελάχιστη απόσταση d ≥ 2t+1.

Κατά την αποκωδικοποίηση χρειαζόμαστε έναν πίνακα ελέγχου υπολοίπων του κώδικα που λέγεται πίνακας ισοτιμίας και συμβολίζεται με H. Μία κωδικολέξη c ικανοποιεί την εξίσωση $cH^T = 0$ όπως φαίνεται παρακάτω:

$$\sum_{i=1}^{n} c_i p_{ij} = 0\ ,\ 1 \le j \le t\ ,$$

$$H = \begin{pmatrix} p_{11} & \cdots & p_{n1} \\ . & . & . \\ p_{1t} & \cdots & p_{nt} \end{pmatrix}$$

Ο πίνακας ελέγχου H χρησιμοποιείται για τον έλεγχο και την εύρεση λαθών (δηλαδή για την αποκωδικοποίηση), αλλά χρειαζόμαστε και ακόμη τον πίνακα γεννήτορα για να κωδικοποιούμε τα μηνύματα σε κωδικολέξεις.

Μια κωδικολέξη προκύπτει ως γινόμενο ενός μηνύματος m με τον πίνακα γεννήτορα G, ο οποίος ικανοποιεί τη σχέση GH^T=0

Δηλαδή η κωδικοποίηση ενός μηνύματος m με τη χρήση ενός Goppa [n, k, d] γίνεται με την κατάτμηση του μηνύματος σε μπλοκ των k χαρακτήρων και πολλαπλασιάζοντας κάθε block, ως διάνυσμα k στοιχείων, με τον πίνακα γεννήτορα G οπότε

$(c_1, c_2, \dots c_n) = (m_1, m_2, \dots m_k) * G$

Το μήνυμα που θα ληφθεί θα είναι της μορφής

$(y_1, y_2, \dots y_n) = (c_1, c_2, \dots c_n) + (e_1, e_2, \dots e_n)$ οπου το λιγοτερο ένα στοιχειο λαθους δεν είναι 0

Για να διορθώσουμε και να βρούμε την πραγματική λέξη $(m_1, m_2, \dots m_k)$θα πρέπει πρώτον να βρεθεί η κωδικολέξη$(c_1, c_2, \dots c_n)$.

Πρώτον βρίσκουμε το διάνυσμα λάθους e, άρα τα στοιχεία του e που δεν είναι 0 και τις αντίστοιχες τιμές τους.

Έπειτα αφού έχουν διορθωθεί τα λάθη βάσει της διαδικασίας που προβλέπεται από τον κώδικα μας, πρέπει να ανακτήσουμε το αρχικό μήνυμα m.

Αφού όμως έχουμε ορίσει ότι ισχύει $(c_1, c_2, \dots c_n) = (m_1, m_2, \dots m_k) *$ G

Αφού έχουν διορθωθεί τα λάθη λαμβάνουμε το αρχικό μήνυμα αν έχουμε τη σωστή κωδική λέξη δηλαδή το $yH^T = 0$ αλλιώς έχει γίνει σφάλμα και με τεχνική αποκωδικοποίησης, αναλόγως του κώδικα, ανακτούμε την αρχική λέξη.

Οπότε έχουμε βρει το αρχικό μήνυμα m. Αυτές είναι γενικές τεχνικές αποκωδικοποίησης κωδίκων. Οι πίνακες G, H επιλέγονται κάθε φορά κατά τρόπο τέτοιο ώστε η αποκωδικοποίηση να είναι εφικτή – αυτό ισχύει και για τους κώδικες Goppa.

Κεφάλαιο 7
Το κρυπτοσυστημα McEliece

7.1 Γενικά στοιχεία

Υπάρχουν αρκετοί λόγοι που η επιλογή για το σύστημα McEliece είναι οι κώδικες Goppa. Έχουν καταρχάς, ένα αρκετά αποδοτικό αλγόριθμο αποκωδικοποίησης σε πολυωνυμικό χρόνο. Αποτελούν ένα κλάδο κωδίκων που είναι δύσκολο να ανακαλύψει κάποιος τις παραμέτρους τους άλλα είναι εύκολο να διαλέξει έναν κώδικα του συνόλου αυτού. Οι γεννήτορες των κωδίκων Goppa είναι σχεδόν τυχαίας δομής πίνακες. Για οποιοδήποτε μήκος n υπάρχουν πολλοί δυνατοί διαφορετικοί κώδικες Goppa. Στην πραγματικότητα ο αριθμός των κωδίκων Goppa αυξάνει

εκθετικά με την αύξηση του μήκους του κώδικα και με την αύξηση του βαθμού του πολυωνύμου γεννήτορα [12].

Το κρυπτοσυστημα McEliece προτάθηκε από τον McEliece το 1978 [15] και είναι παρόμοιο με το παλαιότερο Κρυπτοσυστημα Merkle-Hellman Knapsack αφού και τα δυο αυτά κρυπτοσυστήματα εκμεταλλεύονται μια σχετικά εύκολη περίπτωση ενός προβλήματος NP-complete και το διαμορφώνουν προς την δυσκολότερη μορφή του.

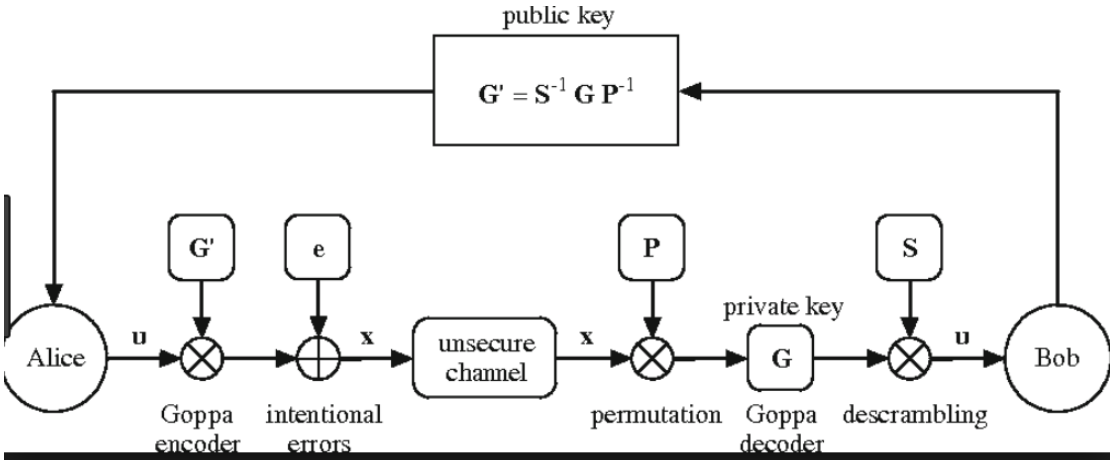
Η τεχνική αποκωδικοποίησης των γραμμικών κωδίκων είναι ένα πρόβλημα NP-complete αν και μόνο αν ο αριθμός των λαθών δεν περιορίζεται. Φυσικά υπάρχουν περιπτώσεις γραμμικών κωδίκων που παρέχουν πολύ γρήγορους αλγορίθμους αποκωδικοποίησης. Η βασική ιδέα του McEliece ήταν να πάρει ένα τέτοιο γραμμικό κώδικα και να τον διαφοροποιήσει έτσι ώστε ο Β, που είναι ο παραλήπτης, να μπορεί να αποκωδικοποιήσει. Από τη μεριά του ο Α, που είναι αποστολέας, μπορεί να χρησιμοποιεί έναν αποδοτικό αλγόριθμο κωδικοποίησης.

Ο McEliece υποστήριξε την υλοποίηση του συστήματος που φέρει το όνομά του με την εφαρμογή κωδίκων Goppa διότι έχουν πολύ αποδοτικό αλγόριθμο αποκωδικοποίησης.

7.2 Το κρυπτοσύστημα McEliece

Το κρυπτοσυστημα McEliece ορίζεται ως εξής:

Επιλέγεται ένα αυθαίρετο πολυώνυμο g(z) ορισμένο σε ένα πεπερασμένο σώμα GF(2^m) το οποίο έχει βαθμό t. Ο κώδικας Goppa ορίζεται από τα L και g(z) και έχει παραμέτρους [n≥ n- mt≥2t + 1]. Υπολογίζεται ο πίνακας γεννήτορας G του κώδικα Goppa διαστάσεων k x n. Επιλέγεται ένας μη μοναδικός πίνακας S, διαστάσεων k x k ο οποίος είναι ψευδοτυχαίος και ένας πίνακας αντιμετάθεσης S διαστάσεων n x n. Υπολογίζουμε το πίνακα G*= $S^{-1}G\,P^{-1}$. Δημοσιοποιούμε τον πίνακα G*, ο οποίος, μαζί με το t, αποτελεί το δημόσιο κλειδί. Τα υπόλοιπα μένουν κρυφά και αποτελούν το ιδιωτικό κλειδί του κρυπτοσυστήματος.



Εικόνα 7.1: Κωδικοποίηση και αποκωδικοποίηση στο σύστημα McEliece [15]

Αν κάποιος θέλει να στείλει ένα μήνυμα, το χωρίζει σε συμβολοσειρές (τμήματα), κάθε ένα εκ των οποίων θα συμβολίζεται κάθε φορά με m και τα οποία περιέχουν μόνο δυαδικά ψηφία ενώ έχουν σταθερό μήκος k. Έπειτα διαλέγει ένα ψευδοτυχαίο διάνυσμα εισαγωγής λαθών e με μήκος n και μέγιστο βάρος t. Δηλαδή σε αυτό το διάνυσμα λάθους δεν μπορεί να υπάρχουν πάνω από t στοιχεία που να έχουν την τιμή 1, με άλλα λόγια $wt(e) \leq t$.

Η κρυπτογράφηση του μηνύματος m είναι y=mG*+e (όπου η πρόσθεση είναι επί δυαδικών διανυσμάτων). Προκύπτει έτσι το διάνυσμα y με n στοιχεία, που είναι μία αλλοιωμένη εκδοχή της κωδικής λέξης mG* του «τυχαίου» κώδικα με πίνακα γεννήτορα G*.

Ο αποστολέας στέλνει αυτό το μήνυμα στο κανάλι επικοινωνίας.

Ο παραλήπτης λαμβάνει το κρυπτογραφημένο y και μπορεί να αποκρυπτογραφήσει με το μυστικό του κλειδί που είναι ο πίνακας P όπως φαίνεται παρακάτω:

$$y' = y\,P^{-1}G + e\,P^{-1} = mSGPP^{-1} + e' = m\,S\,G + e'.$$

Το e' έχει βάρος μικρότερο η ίσο του t, όπως ακριβώς και το e (λόγω του ότι ο πίνακας P είναι πίνακας αντιμετάθεσης, και το ίδιο ισχύει και για τον αντίστροφό του).

Χρησιμοποιώντας τον αλγόριθμο αποκωδικοποίησης Goppa ο παραλήπτης μπορεί να διορθώσει (αποκωδικοποιήσει) το y', το οποίο δεν είναι τίποτα άλλο παρά η αλλοιωμένη εκδοχή της κωδικής

λέξης που αντιστοιχεί στο μήνυμα m'=mS. Με άλλα λόγια, ο παραλήπτης, βρίσκοντας το e' κατά την αποκωδικοποίηση, ανακτά το mS. Όμως γνωρίζει τον S, άρα γνωρίζει και τον αντίστροφο πίνακα S^{-1} . οπότε υπολογίζει το ζητούμενο m=m' S^{-1} .

Για ιστορικούς λόγους αναφέρουμε ότι ο McEliece πρότεινε το μέγεθος του m να είναι 10, καθώς επίσης και t=50 για το πολυώνυμο g(z) .Έτσι οι παράμετροι του κώδικα είναι n=1024 και k≥1024-10*50=525. Να σημειωθεί ότι με τις αναθεωρήσεις ως προς την ασφάλεια του McEliece , η πρόταση στο NIST[21] για την υλοποίηση του McEliece προτείνει τιμές των (n,k,t) τις εξής: (6960, 5413, 119) και (8192, 6528, 128).

Συνοπτικά τα χαρακτηριστικά του κρυπτοσυστήματος [12] φαίνονται στην παρακάτω εικόνα:

PUBLIC:	modified encoding matrix G^*, of size $k \times n$, error-correcting capacity t.
SECRET:	generator polynomial $g(z)$, original encoding matrix G, of size $k \times n$, S and P of size $k \times k$ and $n \times n$ such that $G^* = SGP$.
MESSAGE:	$m \in \{0,1\}^k$.
ENCRYPTION:	$y = mG^* + e$, for $wt(e) \leq t$.
DECRYPTION:	compute $y' = yP^{-1}$, decode y' into $m' = mS$, compute $m'S^{-1} = m$.

Εικόνα 7.2: Παράμετροι στο σύστημα McEliece [12]

7.3 Ασφάλεια του κρυπτοσυστήματος McEliece

Η ασφάλεια του κρυπτοσυστήματος McEliece βασίζεται σε προβλήματα τύπου NP-complete που είναι δύσκολα στον υπολογισμό από άποψη τόσο υπολογιστικών πόρων άλλα και από άποψη χρόνου επίλυσης.

Το ένα πρόβλημα είναι αυτό της εξαντλητικής αναζήτησης στο διανυσματικό χώρο των κλειδιών και το δεύτερο πρόβλημα είναι αυτό της αποκωδικοποίησης βάσει της μέγιστης πιθανοφάνειας (maximum likelihood decoding) – δηλαδή της αποκωδικοποίησης της λαμβανόμενης λέξης βάσει της πιο πιθανής κωδικολέξης που εστάλη. Προφανώς, οι προσπάθειες κρυπτανάλυσης πάνω στο κρυπτοσυστημα McEliece είναι λογικό να στοχεύουν σε κερκόπορτες των δυο αυτών ζητημάτων.

7.3.1 Κρυπτανάλυση τύπου δομικής επίθεσης

Η κρυπτανάλυση τύπου δομικής επίθεσης περιγράφεται από την προσπάθεια να ανακατασκευαστεί ένας αποκωδικοποιητής για τον κώδικα που έχει δημιουργηθεί από δημόσιο κλειδί G*. Εάν συμβεί μια τέτοια επιτυχημένη κρυπταναλυτική τεχνική, τότε αμέσως γίνεται γνωστό το ιδιωτικό κλειδί G και η λογική δημιουργίας του κρυπτοσυστήματος καταρρέει. Η οικογένεια των κωδίκων Goppa μπορεί να χωριστεί σε κλάσεις ισοδυνάμων κωδίκων. Δυο κώδικες μήκους n λέμε ότι είναι ισοδύναμοι όταν υπάρχει η δυνατότητα αντιμετάθεσης n στοιχείων από την οποία να προκύπτει ο δεύτερος κώδικας από τον πρώτο ή το αντίθετο. Έτσι ο κώδικας C' που έχει προκύψει από το γεννήτορα G' και ο κώδικας C του γεννήτορα G ανήκουν στην ίδια ισοδύναμη κλάση. Άρα ο επιτιθέμενος δεν έχει παρά να συγκρίνει έναν κώδικα από κάθε γνωστή κλάση σε σχέση με τον C' ώστε να ευρεθεί ένας ισοδύναμος κώδικας. Δεδομένου ότι οι κλάσεις αυτών των κωδίκων έχουν μεγάλο εύρος, αυτή η διαδικασία είναι ρεαλιστικά αδύνατη με τις υπολογιστικές δυνατότητες που υπάρχουν και ίσως θα υπάρχουν στο μέλλον. Αν μάλιστα χρησιμοποιηθούν οι προτεινόμενες από τον McEliece παράμετροι, οι κώδικες που πρέπει να εξεταστούν είναι 2^{466}. Βέβαια έστω ότι βρίσκεται ένας ισοδύναμος κώδικας, πρέπει να τονισθεί ότι υπάρχει μέθοδος εύρεσης της αντιμετάθεσης.

7.3.2 Επίθεση αποκωδικοποίησης

Επίθεση αποκωδικοποίησης ορίζεται ως η offline αποκωδικοποίηση ενός κρυπτοκειμένου. Φυσικά αυτό δεν εισβάλει στο κρυπτοσυστημα άλλα απλά αν επιτύχει μπορεί να υφαρπάξει μόνο το μήνυμα που κρυπτογραφήθηκε στο συγκεκριμένο κρυπτοκείμενο. Εφόσον οι κώδικες C και C'είναι ισοδύναμοι, έχουν ίδια διορθωτική ικανότητα. Συνεπώς το υπολογιστικό κόστος σε μια τέτοια κρυπταναλυτική επίθεση εξαρτάται μόνο από τις παραμέτρους του C', δηλαδή το μήκος της κωδικής λέξης n, τη διάσταση του κώδικα k και την ικανότητα διόρθωσης t. Με βάση τα κλασσικά μεγέθη του McEliece χρειάζεται 2^{466} υπολογισμούς. Έτσι η κλασσική υλοποίηση του κρυπτοσυστήματος γίνεται ολοένα και λιγότερο ασφαλής. Το πρόβλημα αυτό αντιμετωπίζεται μεγαλώνοντας τα μέγεθος του κλειδιού, αλλά εις βάρος των χρόνων καθυστέρησης

7.3.3 Message-Resend Attack

Ένα από τα κύρια μειονεκτήματα του αρχικού κρυπτοσυστήματος McEliece είναι ότι η κρυπτογράφηση του ιδίου μηνύματος περισσότερες από μία φορές είναι αναγνωρίσιμο και το κείμενο μπορεί να ανακτηθεί από έναν κρυπταναλυτή σε χρόνο ak^3, όπου a είναι μια μικρή σταθερά και k είναι το μέγεθος του μηνύματος του υποκείμενου κώδικα. Το σύστημα αποτυγχάνει επίσης να προστατεύσει οποιαδήποτε μηνύματα που είναι γνωστό ότι έχουν μια γραμμική σχέση μεταξύ τους. Δεδομένου ότι αυτή η επίθεση αποκωδικοποίησης δεν εξαρτάται από τη δομή του κώδικα, αλλάζοντας τον κώδικα ή αυξάνοντας τις τιμές των παραμέτρων δεν αντιμετωπίζεται αποτελεσματικά αυτή την επίθεση[02].

Ο ίδιος ο McEliece πιστεύει ότι η ασφάλεια του κρυπτοσυστήματος του βασίζεται στην επίλυση του προβλήματος αποκωδικοποίησης μιας αυθαίρετης δυαδικής κωδικολέξης και την αντιστοίχιση στην πλησιέστερη λέξη κώδικα στο περιβάλλον ενός αυθαίρετου γραμμικού κώδικα υπό τον περιορισμό ότι η κωδικολέξη έχει βάρος μικρότερο από τη δυνατότητα διόρθωσης σφαλμάτων του κώδικα. Μια πιθανή επίθεση γίνεται ως εξής: Ας υποθέσουμε ότι ο κώδικας Goppa έχει πίνακα γεννήτορα G διαστάσεων (k x n)=(524 × 1024) και πολυώνυμο g(z) βαθμού t=50. Αν ένας εισβολέας μπορούσε να μαντέψει 524 συντεταγμένες(στήλες) μιας κωδικοποιημένης λέξης c που δεν έχουν υποστεί αλλοιώσεις από τον πίνακα σφάλματος e , τότε c* = m(SGP)*. Εάν ο πίνακας SGP* είναι αναστρέψιμος τότε προφανώς έχει βρεθεί η αρχική λέξη m.

Ας δούμε το κόστος αυτής της επίθεσης. Ο επιτιθέμενος πρέπει να μαντέψει 524 στήλες από τις 974 = 1024 - 50, αφού έχουν εισαχθεί 50 σφάλματα στο κρυπτοκείμενο. Ο αριθμός των δυνατών στηλών είναι οι συνδυασμοί 1024 ανά 524 ενώ αν οι στήλες που δεν περιέχουν εισαχθέντα λάθη είναι 974 και γνωστές τότε οι σωστές δυνατότητες υποθέσεων είναι οι συνδυασμοί 974 ανά 524. Άρα για να επιτύχει ο εισβολέας θέλει $\frac{\binom{1024}{524}}{\binom{974}{524}} \approx 1{,}37 * 10^{16}$ προσπαθειες. Συνεπώς, το συνολικό κόστος είναι k* $1{,}37 * 10^{16}$ όπου k το υπολογιστικό κόστος αντιστροφής ενός πίνακα 524 x 524. Αυτή η εξαντλητική μέθοδος δεν είναι εφικτή υπό την τρέχουσα τεχνολογία [02].

Τώρα, ας υποθέσουμε ότι ένα μήνυμα κρυπτογραφείται και αποστέλλεται δύο φορές είτε τυχαία είτε ως αποτέλεσμα δράσης ενός εισβολέα, τότε έχουμε c1 = mSGP + e1 και c2 = mSGP + e2, όπου e1≠e2. Αυτό ονομάζεται κατάσταση επαναποστολής μηνυμάτων(message resend condition), η οποία είναι εύκολα ανιχνεύσιμη με την παρατήρηση του βάρους του αθροίσματος των δύο ciphertexts, c1 και c2. Όταν τα μηνύματα απλού κειμένου είναι πανομοιότυπα, το άθροισμα των c1 και c2 δεν μπορεί να υπερβαίνει, για το παράδειγμά μας όπου t=50, το 100.

7.3.4 Related Message

Η κατάσταση επαναποστολής μηνυμάτων, δηλαδή η περίπτωση μηνυμάτων m1 και m2 που ικανοποιούν τη σχέση m1 + m2 = 0, μπορεί να γενικευθεί σε οποιαδήποτε γραμμική σχέση μεταξύ δυο μηνυμάτων, για παράδειγμα m1 + m2. Ας υποθέσουμε ότι έχουμε δύο μηνύματα m1, m2, με τα αντίστοιχα κρυπτοκείμενά τους c1 και c2, που ικανοποιούν τις σχέσεις, c1 = m1SGP + e1και c2 = m2SGP + e2, τότε c1 + c2 = m1SGP + m2SGP + e1 + e2 = (m1 + m2) SGP + e1 + e2. Το άθροισμα αυτό μπορεί ο επιτιθέμενος να το υπολογίσει εφαρμόζοντας το δημόσιο κλειδί στη γνωστή σχέση m1 + m2. Αφού το c1 + c2 + (m1 + m2) SGP = e1 + e2, η επίθεση μπορεί να γίνει με την ίδια διαδικασία όπως η διαδικασία για το message resend attack. Παρατήρηση: Σημειώστε ότι πρόκειται για επίθεση ανά μήνυμα. Το απλό κείμενο αποκαλύπτεται με πολύ λίγη προσπάθεια αλλά το μυστικό κλειδί παραμένει ασφαλές. Η πρώτη αντίδραση για την αντιμετώπιση αυτής της κρυπταναλυτικής επίθεσης περιλαμβάνει την εισαγωγή ενός στοιχείου ψευδοτυχαιότητας σε ένα μήνυμα πριν κρυπτογραφηθεί. Η ψευδοτυχαιότητα αυτή πρέπει να διασπαρθεί μέσα στο μήνυμα με ένα αρκετά περίπλοκο τρόπο. Ωστόσο, ένα τέτοιο σύστημα με μια τέτοια προσθήκη αρχίζει να γίνεται πιο αργό σε ρυθμό δεδομένων. Το αρχικό κρυπτοσύστημα McEliece πάσχει από αυτού του είδους τις επιθέσεις άλλα η επίθεση αυτή αντιμετωπίστηκε κατά την εισαγωγή του κρυπτοσυστήματος Niederreiter. Στην παραλλαγή αυτή του McEliece εξαλείφεται ο τύπος αυτός της κρυπταναλυτικής επίθεσης.

7.4 Βελτιώσεις του συστήματος McEliece

Έχουν προταθεί κατά καιρούς αρκετές τροποποιήσεις στο κρυπτοσύστημα McEliece χρησιμοποιώντας διαφορετικές οικογένειες κωδίκων Goppa. Οι κώδικες Reed-Solomon (GRS), κώδικες Gabidulin και ReedMuller είναι μεταξύ αυτών που έχουν δοκιμαστεί. Οι τροποποιήσεις με χρήση κωδίκων τύπου Reed-Solomon και κωδίκων Reed-Muller έχουν καταρρεύσει και το σύστημα Gabidulin αποδείχθηκε από τον Gibson να έχει ελάχιστα πλεονεκτήματα σε σχέση με το αρχικό σύστημα McEliece.

Το 2000, ο Pierre Loidreau παρουσίασε μια τροποποίηση για την ενίσχυση του κρυπτοσυστήματος McEliece κατά των επιθέσεων αποκωδικοποίησης με τη χρήση της των

κλάσεων αυτομορφισμού των κωδίκων Goppa και χωρίς αύξηση μεγέθους του δημόσιου κλειδιού. Τα μηνύματα κρυπτογράφησης είναι αλλοιωμένα με μεγαλύτερο αριθμό σφαλμάτων από αυτόν που μπορεί να διορθώσει ο κώδικας. Ωστόσο, λόγω της φύσης του διανύσματος τυχαίου σφάλματος, το κείμενο μπορεί να ανακτηθεί. Το ιδιωτικό κλειδί του τροποποιημένου συστήματος δεν αλλάζει, αλλά το δημόσιο κλειδί περιλαμβάνει ένα πρόσθετο σύνολο που ονομάζεται αποκωδικοποιήσιμο σύνολο t-tower.

7.5 Συμπεράσματα και μελλοντικές χρήσεις

Συμπερασματικά το κρυπτοσύστημα McEliece είναι ασφαλές, αλλά δεν χρησιμοποιείται επί του παρόντος λόγω του μεγέθους του δημοσίου κλειδιού και της μείωσης του ρυθμού δεδομένων κατά 50%. Έχουν βρεθεί μερικές μέθοδοι που αυξάνουν το ρυθμό επεξεργασίας δεδομένων κατά 80% και συμπιέζουν το μεγέθους του ιδιωτικού κλειδιού. Ωστόσο, δεν ξέρουμε πώς αυτές οι μέθοδοι θα λειτουργούσαν με την τροποποίηση του Loidreau.

Πιστεύουμε ότι στο μέλλον το κρυπτοσύστημα McEliece θα γίνει πιο ελκυστικό. Όπως καλυτερεύουν συνεχώς οι τεχνολογικές εξελίξεις, τα μειονεκτήματα του συστήματος θα προσπεραστούν, έτσι ώστε οι κώδικες Goppa να μπορούν να χρησιμοποιηθούν με ακόμα μεγαλύτερες παραμέτρους [17]. Επίσης, καθώς ο αριθμός των κωδίκων Goppa αυξάνεται εκθετικά με το μήκος του κώδικα και τον βαθμό του πολυωνύμου, το κόστος για επιθέσεις τύπου structural attacks κατά του συστήματος θα αυξηθεί.

Το σύστημα McEliece εξάλλου αποτελεί μια εναλλακτική μετακβαντική λύση στα τρέχοντα κρυπτοσυστήματα του δημόσιου κλειδιού, τα οποία βασίζονται σε σημαντικά προβλήματα στη θεωρία των αριθμών. Ήδη αποτελεί έναν από τους αλγορίθμους που έχουν υποβληθεί στο NIST στη διαδικασία επιλογής νέων προτύπων αλγορίθμων. Ένα μειονέκτημα βέβαια είναι η πρόσβαση και ανάγνωση του κλειδιού μεγαλύτερου μεγέθους στο McEliece. Ως εκ τούτου, η συνεχής μελέτη αυτού του συστήματος είναι υψηλής σημασίας για το μέλλον της κρυπτογραφίας.

Κεφάλαιο 8
Υλοποίηση κρυπτοσυστήματος RSA McEliece/AES

Στο σημείο αυτό της εργασίας θα παρουσιάσουμε με όσο το δυνατόν πιο κατανοητό τρόπο την υλοποίηση που κάναμε για να συγκρίνουμε κρυπτοσυστήματα κλασικής κρυπτογραφίας με κρυπτοσυστήματα υποψήφια για την μετά κβαντική εποχή. Απώτερος στόχος είναι η συγκριτική αποτίμηση της απόδοσης του κρυπτοσυστήματος McEliece (αλγόριθμος που φαίνεται ότι θα είναι παρών στη μετακβαντική εποχή) με αυτή του RSA (του πιο διαδεδομένου αντίστοιχου κρυπτοσυστήματος σήμερα, που όμως δεν θα παρέχει ασφάλεια στη μετακβαντική εποχή). Η ανάπτυξη όλων των υλοποιήσεων που αναφέρονται και αιτιολογούνται παρακάτω έγιναν σε περιβάλλον IDE eclipse ® photon σε γλώσσα προγραμματισμού java με βιβλιοθήκες ανοιχτού κώδικα.

8.1 End to end RSA chat

Αρχικά εργαστήκαμε πάνω στην υλοποίηση του κρυπτοσυστήματος δημόσιου κλειδιού RSA από άκρο-σε-άκρο (end-to-end), δηλαδή στην εφαρμογή αυτού του αλγορίθμου τόσο κατά την ανταλλαγή του δημόσιου κλειδιού αλλά και ως αλγόριθμο ο οποίος κρυπτογραφεί και αποκρυπτογραφεί μηνύματα σε μία απλή εφαρμογή συνομιλίας (Chat). Οπότε, σε αυτή την περίπτωση δεν χρησιμοποιήσαμε καθόλου συμμετρική κρυπτογραφία, αλλά όλη η κρυπτογράφηση και αποκρυπτογράφηση έγινε με τον αλγόριθμο RSA.

Το μοντέλο που αναπτύξαμε είναι ένα μοντέλο client-Server και λειτουργεί ως εξής:

Χρησιμοποιούνται 3 java classes, εκ των οποίων μια υλοποιεί τον server, μια τον client, και άλλη μια το πρωτόκολλο RSA.

Η βιβλιοθήκη για την υλοποίηση του RSA είναι java native.

```
import java.security.*
```

Αρχικά ο Server ανοίγει και «ακούει» για συνδέσεις.

```
ChatHistory.setText("Waiting for Client");
  conn = server.accept();
```

Έπειτα παράγει το ζεύγος ιδιωτικού και δημόσιου κλειδιού RSA, με κλήση της αντίστοιχης συνάρτησης:

```
KeyPair keypair = createkeypair();
```

που καλεί την κλάση RSA:

public static KeyPair buildKeyPair() **throws** NoSuchAlgorithmException {

```
    final int keySize = 2048;
    KeyPairGenerator = KeyPairGenerator.getInstance("RSA");
    keyPairGenerator.initialize(keySize);
    return keyPairGenerator.genKeyPair();
  }
```

Και επιστρέφει το ζεύγος κλειδιών του Server:

PrivateKey Serverprivate = keypair.getPrivate();

Έπειτα ανοίγει ο Client και ακολουθεί τον ίδιο δρόμο παράγοντας το ζεύγος ιδιωτικού-δημοσίου RSA κλειδιού με κλήση στην κλάση RSA.java όπως και ο Server.

Μετά στέλνεται από τον client το δημόσιο κλειδί του στον server ο οποίος και το αποθηκεύει σε αρχείο.

Χρησιμοποιείται socket και αποστέλλονται τα κλειδιά ως αντικείμενα και όχι ως δυαδικός πίνακας τύπου byte ως εξής:

```
ObjectOutputStream
Serverpubkeystream=new ObjectOutputStream(s.getOutputStream());
```

Το ίδιο γινεται και από τον server στον client με την ίδια μέθοδο αποστολής και αποθήκευσης. Στην ουσία έχουμε χρησιμοποιήσει 2 socket, ένα για την αποστολή δημοσίου κλειδιού στον Client και ένα για την αποστολή δημοσίου κλειδιού στον Server. Έτσι και οι δύο γνωρίζουν το Δημόσιο κλειδί της άλλης οντότητας οπότε και μπορεί να ξεκινήσει η διαδικασία ανταλλαγής γραπτών μηνυμάτων μέσω της socket port 2000. Στην παρούσα διαδικασία δεν χρησιμοποιήσαμε ψηφιακά πιστοποιητικά των χρηστών για λόγους απλότητας.

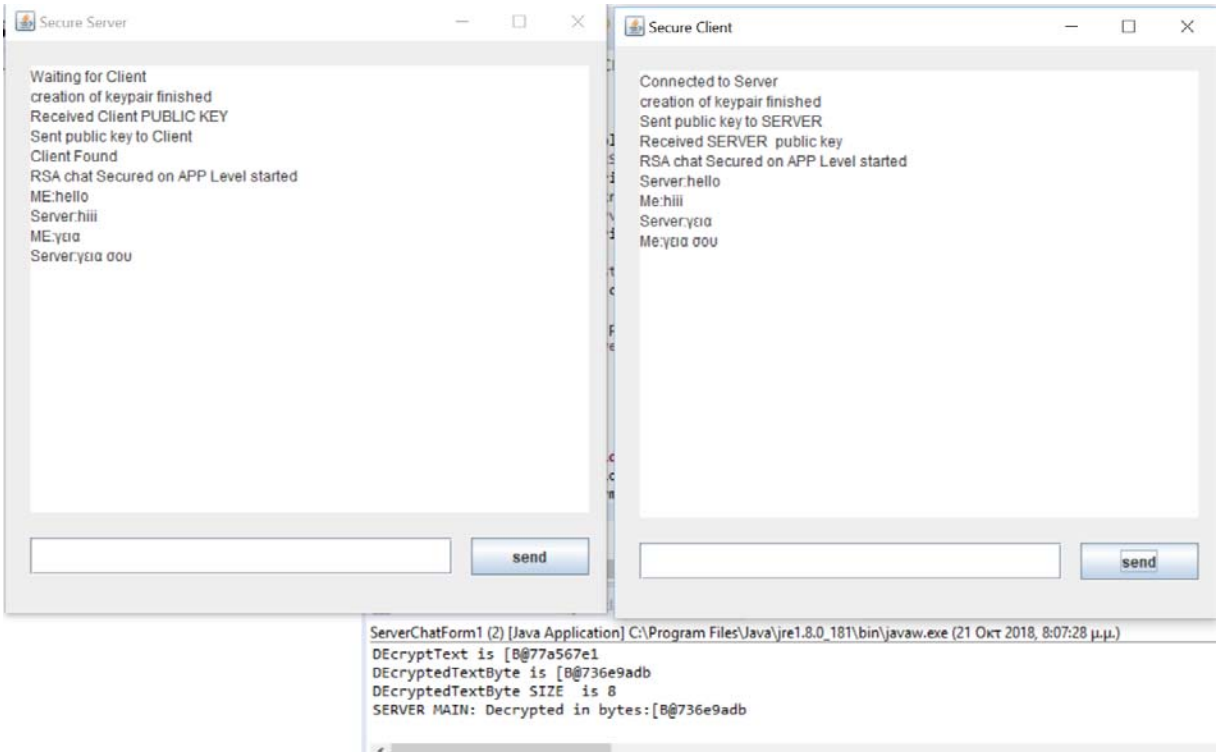
Ο client και ο Server έχουν υλοποιηθεί ώστε να φαίνονται σε ένα περιβάλλον γραφικών και διαδραστικό, στο οποίο ο χρήστης μπορεί να εισάγει κείμενο και με το κουμπί αποστολής να το αποστείλει είτε προς τη μία είτε προς την άλλη μεριά. Όταν θέλει κάποιος εκ των δύο να στείλει ένα μήνυμα τότε αυτό που συμβαίνει στο «παρασκήνιο» είναι ότι το μήνυμα ως απλό κείμενο φαίνεται στο διαδραστικό περιβάλλον του αποστολέα ενώ για να αποσταλεί στον παραλήπτη κρυπτογραφείται με το Δημόσιο κλειδί του παραλήπτη με κλήση της αντίστοιχης κλάσης RSA:

```
public static byte[] encryptText(PublicKey , byte [] message ) throws Exception {
    Cipher = Cipher.getInstance("RSA");
    System.out.println("encryptText is "+ message);
    cipher.init(Cipher.ENCRYPT_MODE, publicKey);
    System.out.println("Cipher inited ");
    byte[] textbyte = cipher.doFinal(message);
    System.out.println("encryptedTextByte is "+ textbyte);
    System.out.println("encryptedTextByte SIZE is "+ textbyte.length);
    return textbyte;
     }
```

Στη συνέχεια ο παραλήπτης το παραλαμβάνει και το αποκρυπτογραφεί χρησιμοποιώντας το ιδιωτικό κλειδί του, πάλι με κλήση στην κλάση RSA:

```
  public static  byte[] decryptText(PrivateKey , byte [] message) throws Exception
{
    Cipher = Cipher.getInstance("RSA");
    cipher.init(Cipher.DECRYPT_MODE,privateKey );
    System.out.println("DEcryptText is "+ message);
    byte[] textbyte = cipher.doFinal(message);
    System.out.println("DEcryptedTextByte is "+ textbyte);
    System.out.println("DEcryptedTextByte SIZE is "+ textbyte.length);
    return textbyte;
  }
```

.Στην παρούσα εργασία δε μας ενδιαφέρει τόσο η σύγκριση ως προς το δίκτυο, οπότε προτιμούμε να «εκτελέσουμε» τον εξυπηρετητή (server) αλλά και τον πελάτη (Client) στο ίδιο μηχάνημα. Παρακάτω ακολουθεί στιγμιότυπο οθόνης από την υλοποίηση και την επίτευξη επικοινωνίας, όπως αυτό φαίνεται.



Εικόνα 8.1: Υλοποίηση end-to-end RSA chat όπως περιγράφεται στο κεφάλαιο 8. 1

8.2 End to end McEliece chat

Βασισμένοι σε αυτή την υλοποίηση που προαναφέρθηκε εργαστήκαμε προς την κατεύθυνση της υλοποίησης end-to-end του ίδιου μοντέλου συνομιλίας αλλά πλέον χρησιμοποιώντας το μετακβαντικό αλγόριθμο McEliece.

Και πάλι έχουμε ακριβώς την ίδια υλοποίηση όπως αναφέρθηκε παραπάνω αλλά αυτή τη φορά χρησιμοποιούμε την βιβλιοθήκη FlexiProvider για την παραγωγή την αποθήκευση και τον χειρισμό των κλειδιών του κρυπτοσυστήματος McEliece.

```
import de.flexiprovider.api.exceptions.InvalidKeySpecException;
import de.flexiprovider.api.keys.KeySpec;
import de.flexiprovider.pki.PKCS8EncodedKeySpec;
import de.flexiprovider.pqc.FlexiPQCProvider;
```

Το μοντέλο που αναπτύξαμε είναι ένα μοντέλο client-Server και λειτουργεί ως εξής :

Χρησιμοποιούνται 3 java classes, εκ των οποίων μια υλοποιεί τον server, μια τον client και μία τον αλγόριθμο McEliece.

Ανοίγει ο εξυπηρετητής και παράγει το ζεύγος Δημοσίου και ιδιωτικού κλειδιού McEliece με κλήση στην αντίστοιχη κλάση:

```
public static java.security.KeyPair generateKeyPair(){

  int KeySize = 1024;
  Security.addProvider(new FlexiPQCProvider());

  try
  {  KeyPairGenerator kpg = KeyPairGenerator.getInstance("McEliece","FlexiPQC");
       kpg.initialize(KeySize);
    java.security.KeyPair keypair = kpg.generateKeyPair();
             return keypair;
  } catch (Exception e) {
    System.out.println(e.toString());
  }
       return null;

  }
```

Είναι έτοιμος και «ακούει» για συνδέσεις chat.

Έπειτα ανοίγουμε το client το οποίο και πάλι παράγει το δικό του ζεύγους κλειδιών McEliece κατά τον ίδιο τρόπο.

Ακολούθως λαμβάνει χώρα η ανταλλαγή δημοσίων κλειδιών με sockets σε επίπεδο bytes και όχι αντικειμενοστραφώς(Αυτό επηρεάζει τη σύγκριση στους χρόνους ενδεχομένως) όπως στην περίπτωση του end to end RSA chat. Αλλά ως εξής:

```
File clientpubkeyfile = new File ("C:/Client/Clientpublic.key");
    byte [] mybytearray = new byte [(int)clientpubkeyfile. length ()];
    FileInputStream fis = new FileInputStream(clientpubkeyfile);
    BufferedInputStream bis = new BufferedInputStream(fis);
    bis.read(mybytearray, 0, mybytearray.length);
    OutputStream os = clientSocket.getOutputStream();
    os.write(mybytearray, 0, mybytearray.length);
    System.out.println("Sending " + "C:/Client/Clientpublic.key" + "(" +
mybytearray.length + " bytes)");
             os.flush();
             if (bis != null) bis.close();
    if (os != null) os.close();
    if (clientSocket!=null) clientSocket.close();
```

Εδώ παρατηρούμε ότι όλα τα κλειδιά γράφονται σε αρχεία σε σχετικούς φακέλους με ονομασία που μαρτυρεί την προέλευσή τους.

Έτσι και οι δύο πλευρές γνωρίζουν το δημόσια κλειδιά τους και συνδέεται ο client στο Server με την δυνατότητα αποστολής κρυπτογραφημένου κειμένου.

Ο αποστολέας (όποιος και να είναι εκατέρωθεν) κρυπτογραφεί ή αλλιώς κωδικοποιεί το καθαρό κείμενο το οποίο θέλει να στείλει, με το Δημόσιο κλειδί του παραλήπτη. Εδώ καλείται, κατά το action SEND, η συνάρτηση keyreader που έχει φτιαχτεί για να διαβάζει και να αναδημιουργεί το κλειδί McEliece από το αρχείο που αυτό έχει αποθηκευτεί.

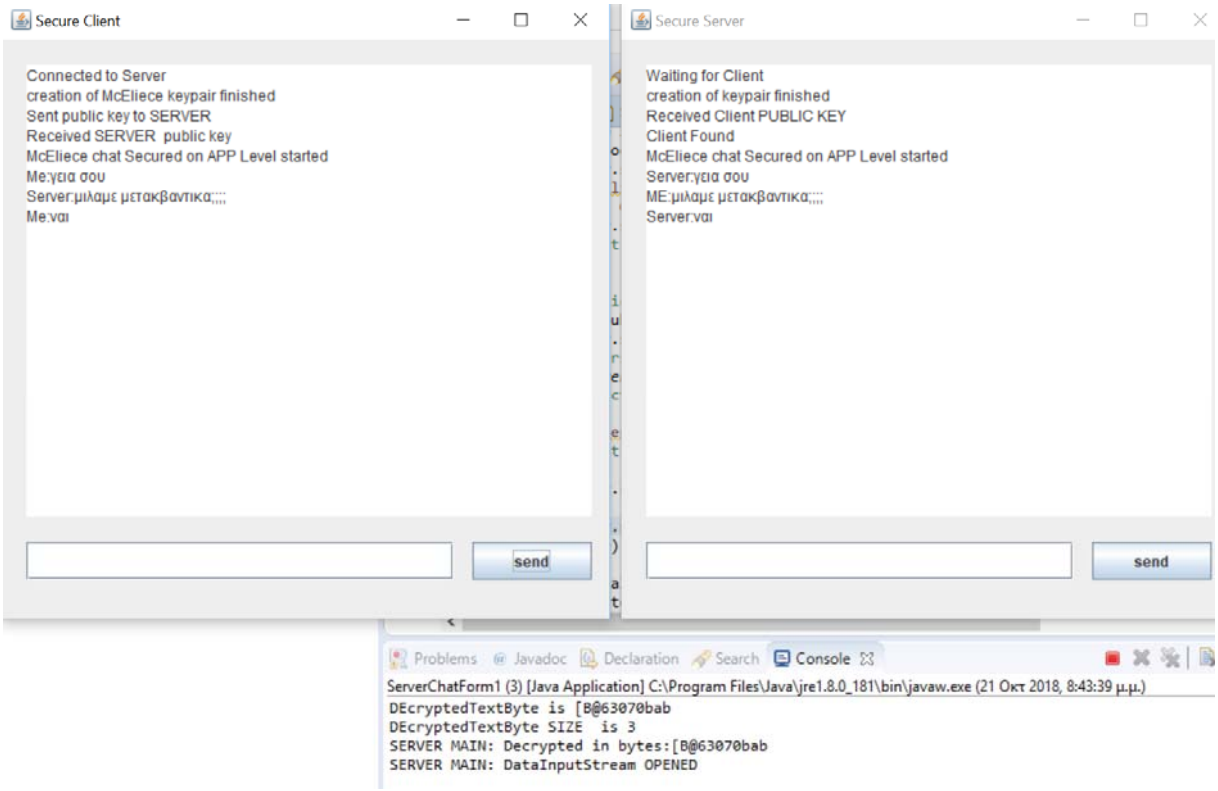
```
      public static java.security.PublicKey PublicKeyReader(String filename)
throws IOException, InvalidKeyException, NoSuchAlgorithmException,
NoSuchProviderException, java.security.spec.InvalidKeySpecException
      {
byte[] keyBytes = Files.readAllBytes(new File(filename).toPath());
               KeySpec publicKeySpec = new X509EncodedKeySpec(keyBytes);
               KeyFactory = java.security.KeyFactory.getInstance("McEliece",
"FlexiPQC");
               PublicKey = keyFactory.generatePublic(publicKeySpec);
return publicKey;

 }
```

Ο παραλήπτης αποκωδικοποιεί με το ιδιωτικό του κλειδί. Όποτε έχουμε ασφάλεια με μετακβαντικό κρυπτογραφικό αλγόριθμο σε επίπεδο εφαρμογής.

Υπήρξε αρκετά μεγάλη δυσκολία ως προς την εύρεση του κατάλληλου σχήματος και της κατάλληλης βιβλιοθήκης για την αποθήκευση και ανάγνωση από αρχείο, δηλαδή την ανάκτηση του κλειδιού από το αρχείο και την σχηματοποίηση του σε object. Όπως προαναφέρθηκε χρησιμοποιήθηκε η βιβλιοθήκη ανοικτού κώδικα FlexiProvider. Ωστόσο υπάρχουν κι άλλες βιβλιοθήκες, όπως η bouncy castle και οι βιβλιοθήκες της Java οι οποίες όμως δεν υποστηρίζουν το κρυπτοσύστημα αυτό.

Ακολουθεί ένα στιγμιότυπο οθόνης του McEliece end to end chat



Εικόνα 8.2: Υλοποίηση end-to-end McEliece chat όπως περιγράφεται στο κεφάλαιο 8.2

Μέχρι τώρα είδαμε στιγμιότυπα εικόνας για τα σχήματα που προαναφέρθηκαν με τη χρήση μόνο RSA και μόνο McEliece, για πλήρη κρυπτογράφηση με αυτά από άκρο σε άκρο (end-to-end). Φυσικά, δεν θα μπορούσαμε παρά να προχωρήσουμε σε μια πιο ολοκληρωμένη παρουσίαση και σύγκριση των κρυπτοσυστημάτων δημόσιου κλειδιού με την εισαγωγή ενός συμμετρικού αλγορίθμου όπως ο AES με κλειδί μήκους 256 bit (μέγεθος το οποίο θα χρησιμοποιείται και στη μετακβαντική εποχή).

8.3 RSA/AES Chat

Σε αυτήν την περίπτωση αλλάζουν κυρίως οι συναρτήσεις κωδικοποίησης και αποκωδικοποίησης κειμένου, διότι πλέον δε γίνεται χρήση ενός αλγορίθμου αλλά δύο αλγορίθμων: ο ένας για την κρυπτογραφία δημοσίου κλειδιού και ο δεύτερος για την συμμετρική κρυπτογράφηση κειμένου. Προστίθεται στο Server η δυνατότητα παραγωγής συμμετρικού κλειδιού AES, ενώ επίσης προσθέτουμε μία συνάρτηση μεταφοράς του συμμετρικού κλειδιού με βάση το κάθε κρυπτοσύστημα που χρησιμοποιούμε - δηλαδή McEliece ή RSA.

Χρησιμοποιούνται 4 java classes, εκ των οποίων μια υλοποιεί τον server, μια τον client, μια τον AES και άλλη μια το πρωτόκολλο RSA (Παράρτημα Α.1).

Στην περίπτωση της υλοποίησης με RSA – AES έχουμε τα εξής βήματα.

Ανοίγει ο Server και παράγει το ζευγάρι κλειδιών RSA. Έπειτα αναμένει για σύνδεση με τον client:

```
public static KeyPair buildKeyPair() throws NoSuchAlgorithmException
  {
    final int keySize = 2048;
    KeyPairGenerator = KeyPairGenerator.getInstance("RSA");
    keyPairGenerator.initialize(keySize);
    return keyPairGenerator.genKeyPair();
 }
```

Ανοίγει ο client και παράγει κατά τον ίδιο τρόπο το RSA το ζεύγος ιδιωτικού και δημόσιου κλειδιού του.

Στη συνέχεια γίνονται οι διαδικασίες ανταλλαγής των δημοσίων κλειδιών μέσω sockets, με την ιδιότητα που προσφέρει η java για την ανταλλαγή αντικειμένων object όπως προαναφέρθηκε στο 8.1.

Έπειτα ο Server παράγει ένα μυστικό συμμετρικό κλειδί βάσει του αλγορίθμου AES:

```
public static SecretKey buildKey() throws NoSuchAlgorithmException {
       KeyGenerator keyGen = KeyGenerator.getInstance("AES");
       keyGen.init(256);
       SecretKey = keyGen.generateKey();
       return secretKey;
        }
```

Το οποίο κρυπτογραφείται με το Δημόσιο RSA κλειδί του client και το αποστέλλει.

```
public static byte[] encryptAES(PublicKey publicKey , byte [] AESkeybytes ) throws
Exception {
    Cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    cipher.init(Cipher.ENCRYPT_MODE, publicKey);
    byte[] encryptedAESkeybytes = cipher.doFinal(AESkeybytes);
    return encryptedAESkeybytes;

 }
```

Ο client το αποκρυπτογραφεί με τη χρήση της συνάρτησης αποκρυπτογράφησης του RSA.

```
public static  SecretKey decryptAES(PrivateKey , byte [] encryptedAESkeybytes)
throws Exception {
    Cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    cipher.init(Cipher.DECRYPT_MODE,privateKey );
    byte[] keybytesALL = cipher.doFinal(encryptedAESkeybytes);
    byte[] keybytes= Arrays.copyOfRange(keybytesALL, 0, 32);
    SecretKey key = new SecretKeySpec ( keybytes, "AES" );
    return key;
 }
```

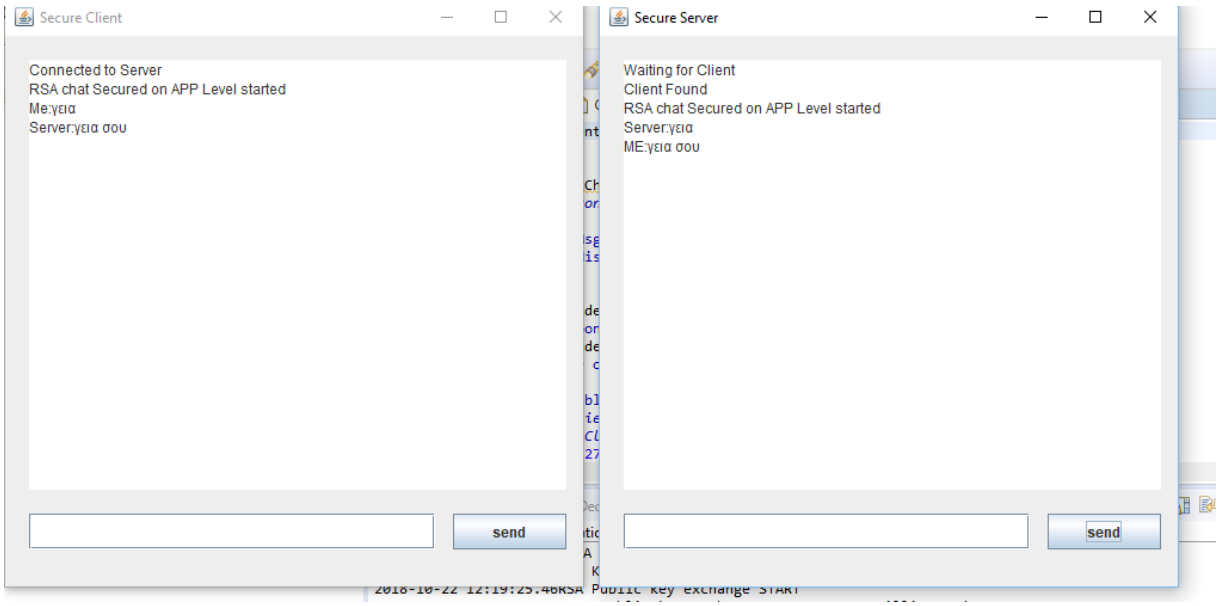
Τώρα και τα δύο μέρη γνωρίζουν το μυστικό ή αλλιώς το συμμετρικό κλειδί. Έτσι μπορεί να αρχίσει η διαδικασία συζήτησης στο γραφικό περιβάλλον όπως προαναφέραμε. Κάθε φορά ως γνωστόν, βάσει των αρχών της συμμετρικής κρυπτογραφίας κάθε μήνυμα για να αποσταλεί κρυπτογραφείται με το συμμετρικό κλειδί AES καλώντας τη συνάρτηση κρυπτογράφησης AES.

```
public static byte[] encryptText(SecretKey , String message ) throws Exception
```

Έπειτα ο παραλήπτης του αποκρυπτογραφεί με το ίδιο μυστικό κλειδί καλώντας τη συνάρτηση αποκρυπτογράφησης.

```
public static String decryptText(SecretKey , byte [] ciphertext)
```

Ακολουθεί ένα στιγμιότυπο οθόνης του RSA/AES chat:



Εικόνα 8.3: Υλοποίηση RSA/AES chat όπως περιγράφεται στο κεφάλαιο 8.3

8.4 McEliece/AES Chat

Στη συνέχεια θα συζητήσουμε την ίδια υλοποίηση με κρυπτοσύστημα McEliece και συμμετρικό κλειδί πάλι τύπου AES. Κινούμενοι στα ίδια πλαίσια και με τη χρήση της γνωστής βιβλιοθήκης FlexiProvider εργαστήκαμε κατά τον ακόλουθο τρόπο.

Χρησιμοποιούνται 4 java classes, εκ των οποίων μια υλοποιεί τον server, μια τον client, μια τον AES και άλλη μια το πρωτόκολλο McEliece (Παράρτημα Α.2).

Αρχικά ανοίγουμε το Server, ο οποίος παράγει το ζεύγος κλειδιών του αλγορίθμου McEliece. Έπειτα περιμένει για σύνδεση με τον Client.

Ανοίγουμε το client το οποίο και αυτό παράγει το δικό του ζεύγος κλειδιών McEliece.

Αμέσως μετά αρχίζει διακίνηση των δημοσίων κλειδιών κατά τα γνωστά (8.2) και προτιμήσαμε την εγγραφή των κλειδιών τόσο του Server όσο και το client σε αρχείο και την αποστολή με hardcoded binary δηλαδή δυαδικό πίνακα bytes [].

Έτσι λοιπόν στέλνεται μέσω socket ως δυαδικός πίνακας το Δημόσιο κλειδί από το ένα μέρος στο άλλο μέρος και αντίστροφα.

Αμέσως το κάθε μέρος εγγράφει το Δημόσιο κλειδί του άλλου μέρους σε αρχείο ώστε να μπορεί να το ανακτήσει κάθε στιγμή.

Ευθύς αμέσως ο Server δημιουργεί ένα συμμετρικό κλειδί AES. Έπειτα το κρυπτογραφεί με το Δημόσιο κλειδί McEliece του client

```
      public static byte[] encryptText(java.security.PublicKey publicKey, byte[]
message ) throws IOException, InvalidKeyException, NoSuchAlgorithmException,
NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException
      {
 System.out.println("Encrypting text");
             Cipher = Cipher.getInstance("McEliecePKCS"); // Obtain  McEliecePKC
Cipher Object
             cipher.init(Cipher.ENCRYPT_MODE, publicKey); // Init the cipher
             byte[] ciphertextBytes = cipher.doFinal(message); // encrypt
             return ciphertextBytes;

 }
```

και το στέλνει σε αυτόν κρυπτογραφημένο σε μορφή πίνακα δυαδικών στοιχείων bytes[].

Ο client παραλαμβάνει αυτόν τον πίνακα δυαδικών στοιχείων και τον αποθηκεύει σε αρχείο. Τέλος χρησιμοποιεί το ιδιωτικό του κλειδί McEliece και αποκρυπτογραφεί το κρυπτογραφημένο συμμετρικό κλειδί.

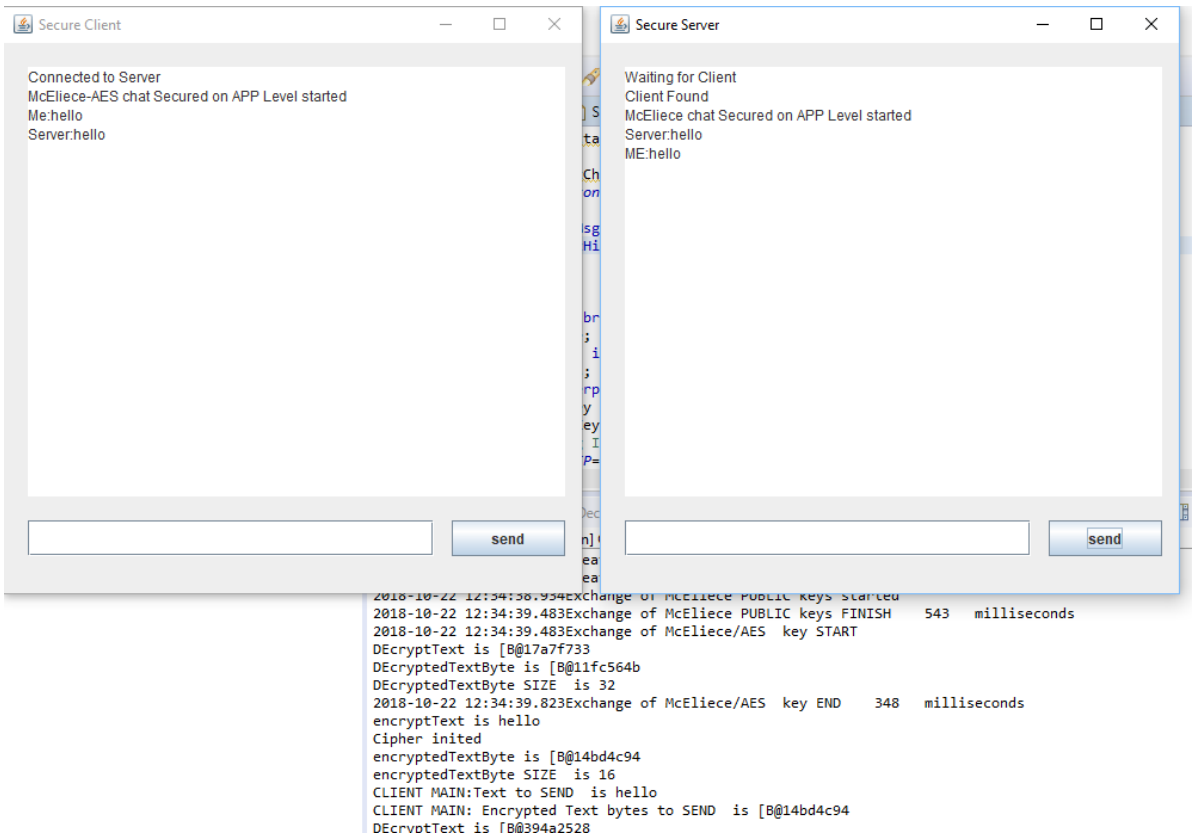
```
public static  byte[] decryptText(java.security.PrivateKey privateKey , byte []
message) throws Exception {

           Cipher = Cipher.getInstance("McEliecePKCS");
           cipher.init(Cipher.DECRYPT_MODE, privateKey);

           byte[] textbyte = cipher.doFinal(message);

           System.out.println("DEcryptedTextByte SIZE is "+ textbyte.length);
           return textbyte;
       }
```

Το συμμετρικό κλειδί το αποθηκεύει και αυτό σε αρχείο. Από αυτό το σημείο και μετά ξεκινάει η διαδικασία αποστολής και λήψης μηνυμάτων βάση των γνωστών εφαρμοσμένων παραπάνω περί συμμετρικού κλειδιού.



Εικόνα 8.4: Υλοποίηση McEliece/AES chat όπως περιγράφεται στο κεφάλαιο 8.4

8.5 Μετρήσεις και αποτελέσματα

Το μέγεθος κλειδιού για τον RSA επιλέχθηκε να είναι το 2048 καθώς αυτό αποτελεί ένα τυπικό μήκος κλειδιού σε διάφορες εφαρμογές. Επιπλέον είναι ίδιου επιπέδου ασφάλειας με τον McEliece 1024 [15],[17]. Παρόλα αυτά και πάλι φαίνεται σαφής καθυστέρηση στον McEliece καθότι είναι αλγόριθμος με χρήση κώδικα και όχι παραγοντοποίησης μεγάλων αριθμών. Για την αποτίμηση της αποδοτικότητας αλλά και τη σύγκριση των αλγορίθμων McEliece και RSA όπως αυτά υλοποιήθηκαν στη java μετρήθηκαν τα εξής:

i. χρόνος δημιουργίας ζεύγους κλειδιού

ii. χρόνος ανταλλαγής δημόσιου κλειδιού (όπου συμπεριλαμβάνεται και ο χρόνος μετάδοσης)

iii. χρόνος ανταλλαγής συμμετρικού κλειδιού. (όπου συμπεριλαμβάνεται και ο χρόνος μετάδοσης και κωδικοποίησης-αποκωδικοποίησης)

Δε μας ενδιαφέρει η μέτρηση χρόνου αποστολής και λήψης μηνυμάτων γιατί αυτή κατά κανόνα γίνεται με κρυπτογράφηση συμμετρικού κλειδιού, κάτι που είναι εκτός των απαιτήσεων της παρούσης εργασίας καθώς δεν μελετούμε τα συστήματα συμμετρικού κλειδιού. Να σημειωθεί παρόλα αυτά πως με αύξηση του μήκους του κλειδιού οι συμμετρικοί αλγόριθμοι κρυπτογράφησης είναι ανθεκτικοί στη μετακβαντική εποχή. Την στιγμή των δοκιμών και στα δύο συστήματα, δεν έτρεχαν διεργασίες στο προσκήνιο ή παρασκήνιο, παρά μόνον ό,τι προβλέπεται από το λειτουργικό σύστημα. Δοκιμάσαμε την υλοποίηση μας σε Windows 7 και Windows 10 συστήματα με τα κάτωθι χαρακτηριστικά:

Συστήματα /χαρακτηριστικά	Windows 7	Windows 10 Enterprise
Λειτουργικό	Microsoft Windows 7 Professional 64-bit	Microsoft Windows 10 (10.0) Enterprise Edition 64-bit
CPU	Intel(R) Core (TM) i5-3570K CPU @ 3.40GHz	Intel(R) Core (TM) i5-8350U CPU @ 1.70GHz
RAM	DDR3 8Gb	DDR4 8Gb
MotherBoard	Intel Ivy Bridge rev. 09 /Intel Z77 rev. 04	Intel Kaby Lake rev. 08 Intel Coffee Lake-U/Y PCH rev. 21
Δίσκος	250 GB SSD	250 GB SSD
Java version	java version "1.8.0_191"	java version "1.8.0_191"

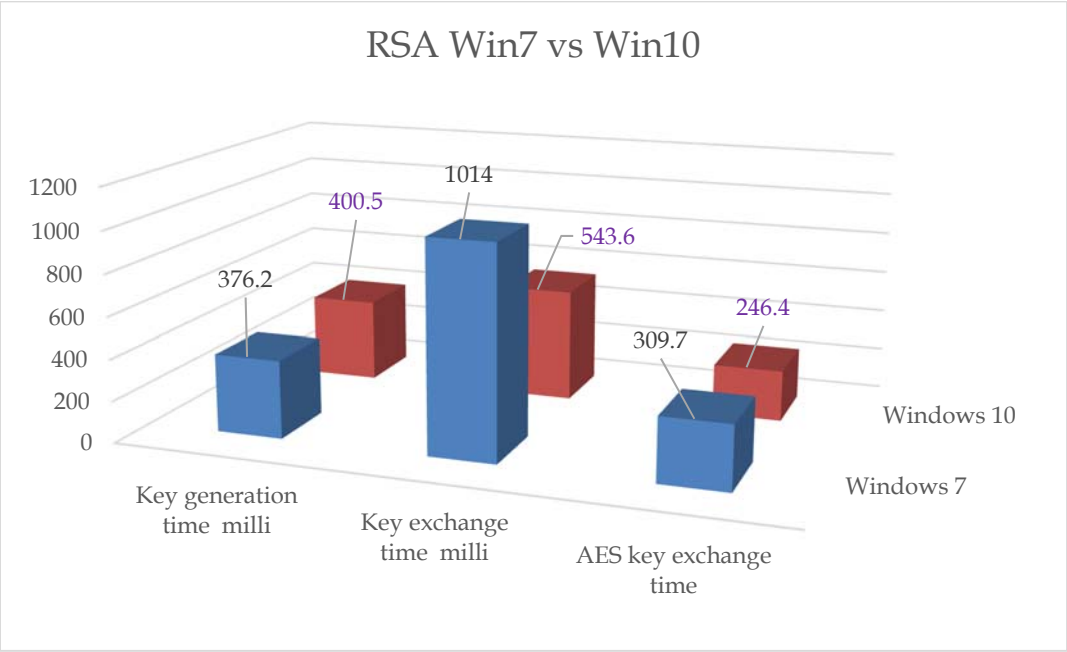
Πίνακας 8.1: Στον πίνακα φαίνονται τα χαρακτηριστικά των 2 συστημάτων Windows

Παρακάτω φαίνονται τα συγκριτικά αποτελέσματα σε χρόνο και σε κλάσμα $\frac{t_{McEliece}}{t_{RSA}}$, των αντίστοιχων χρόνων που χρειάστηκαν στα μηχανήματα για τον αλγόριθμο RSA και τα δεδομένα

που προέκυψαν. Προχωρήσαμε σε 10 μετρήσεις(Παράρτημα Β) για κάθε μία από τις δύο υλοποιήσεις ώστε να εξάγουμε μέσες τιμές όπως φαίνονται παρακάτω. Προφανώς το σύστημα που χρησιμοποιεί Windows 10 έχει καλύτερο επεξεργαστή και μητρική πλακέτα όποτε περιμένουμε να φανεί κάποια διαφορά.

Σε αυτό το σημείο θα πρέπει να σημειώσουμε πως υπήρξε μια διαφορά στην υλοποίηση του RSA έναντι του McEliece λόγω χρηστικότητας βιβλιοθηκών java. Για την ακρίβεια, η υλοποίηση RSA χρησιμοποιεί μεταφορά αντικειμένων μέσω sockets. Δηλαδή τα αντικείμενα είναι διαθέσιμα και προσβάσιμα στην μνήμη της διεργασίας που εκτελεί ο κώδικας όμως η υλοποίηση αυτή αυξάνει τους χρόνους κατάτι, καθώς η μεταφορά αντικειμένων λαμβάνει χώρα σε μεγαλύτερο χρόνο σε σχέση με τη μεταφορά bytes. Αντιθέτως στην υλοποίηση McEliece χρησιμοποιήσαμε μεταφορά σε δυαδική μορφή, οπότε έχουμε αντίστοιχα λίγο καλύτερη απόδοση εις βάρος της υλοποίησης του προγράμματος με RSA. Αυτό είναι σημαντικό να το έχουμε κατά νου στη συγκριτική αποτίμηση που ακολουθεί, γιατί θα δούμε ότι παρά το ότι η υλοποίηση «επιβαρύνει» τον RSA, εξακολουθεί να είναι πολύ πιο αποδοτικός από τον αλγόριθμο McEliece. Όμως σε κάθε περίπτωση ο χρόνος ανταλλαγης δημόσιου κλειδιού είναι ευθέως ανάλογος του μεγέθους κλειδιού και εξαρτάται από την υλοποίηση. Δεν σχετίζεται ωστόσο, με τη μέθοδο κωδικοποίησης και αποκωδικοποίησης στο κάθε κρυπτοσύστημα. Το γεγονός της αναφοράς σε αυτό το χρόνο γίνεται για λόγους **κυρίως πληρότητας** στην επεξήγηση της υλοποίησης στην κάθε περίπτωση αλγορίθμου.

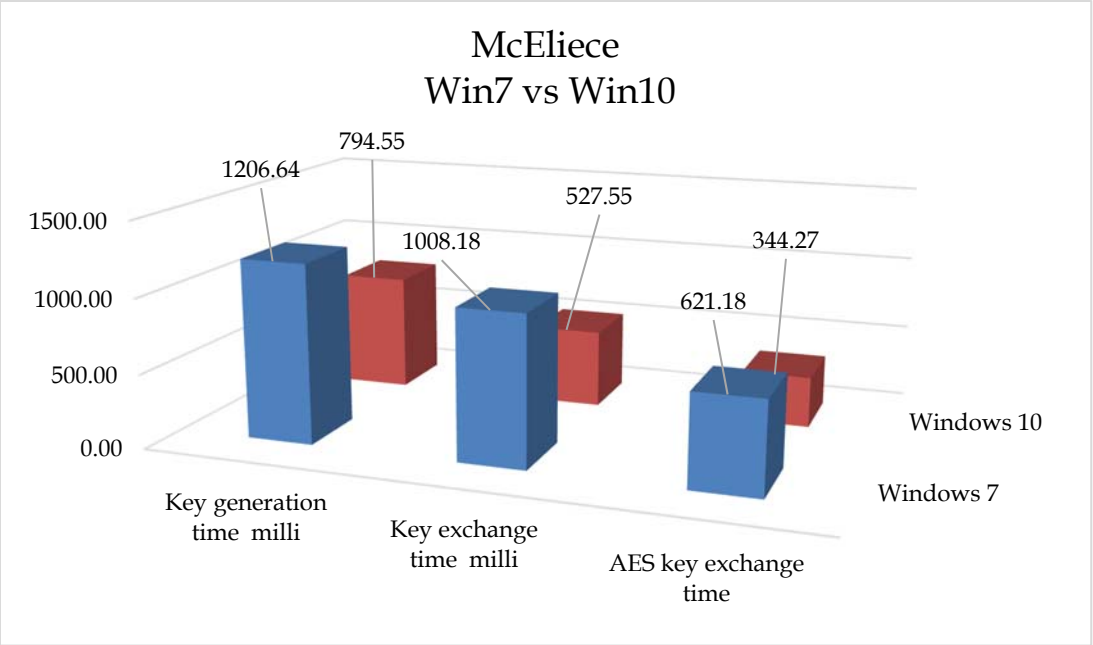
8.5.1 Σύγκριση υλοποίησης RSA στα δύο συστήματα Windows



Σχήμα 8.1: Χρόνοι σε milli seconds για την εφαρμογή RSA/AES chat στα 2 συστήματα.

Ως προς τις μετρήσεις των χρονών για την υλοποίηση RSA-2048 παρατηρούμε ότι χρειάζονται από 376 (Windows 7) έως 400 ms (Windows 10) για τη δημιουργία του ζεύγους κλειδιών. Τα ζεύγη θα μπορούσαμε να υποθέσουμε ότι είναι μόνιμα. Δηλαδή μια οντότητα κρατά πάντα ίδιο ζεύγος γιατί δημοσιοποιεί το κλειδί της οπότε δεν έχει λόγο να αλλάξει ζεύγος. Αυτό σημαίνει ότι είναι πραγματικά ενδιαφέρον να γνωρίζουμε το χρόνο παραγωγής ζεύγους κλειδιών αλλά μόνο για λόγους αναφοράς και όχι στην πράξη. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζονται από 550 (Windows 10) έως 1000 ms (Windows 7) . Όπως προ είπαμε, η υλοποίηση ειδικά για τον RSA, έγινε με μεταφορά αντικειμένων (key objects -java) κάτι που προφανώς περιλαμβάνει serialize και de serialize. Για την αποστολή του μυστικού κλειδιού χρειάστηκαν από 246 (Windows 10) έως 309 ms(Windows 7). Παρατηρούμε ότι και στα 2 συστήματα οι χρόνοι είναι της τάξεως των 300 έως 1000 ms. Στο σύστημα windows10 που έχει σαφώς καλύτερα χαρακτηριστικά, σημειώνουμε και μικρότερους χρόνους.

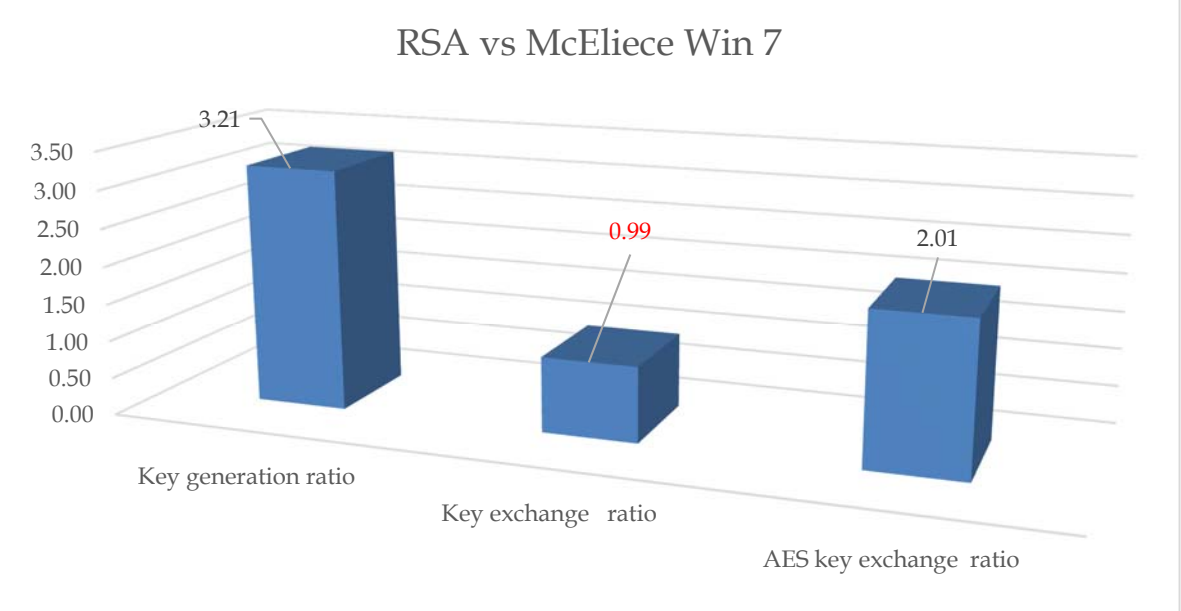
8.5.2 Σύγκριση υλοποίησης McEliece στα δύο συστήματα Windows



Σχήμα 8.2: Χρόνοι σε milli seconds για την εφαρμογή McEliece/AES chat στα 2 συστήματα

Ως προς τις μετρήσεις των χρονών για την υλοποίηση McEliece-1024 παρατηρούμε ότι χρειάζονται από 794(Windows 10) έως 1206 ms (Windows 7) για τη δημιουργία του ζεύγους κλειδιών. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζονται από 527 (Windows 10) έως 1008 ms (Windows 7) . Όπως προαναφέρθηκε, η υλοποίηση για τον McEliece-1024 έγινε με μεταφορά σε σειριακή μορφή Byte[]. Για την αποστολή του μυστικού κλειδιού χρειάστηκαν από 344 (Windows 10) έως 621 ms (Windows 7). Παρατηρούμε ότι και στα 2 συστήματα οι χρόνοι είναι της τάξεως των 350 έως 1200 ms. Στο σύστημα windows10 που έχει σαφώς καλύτερα χαρακτηριστικά, σημειώνουμε και μικρότερους χρόνους.

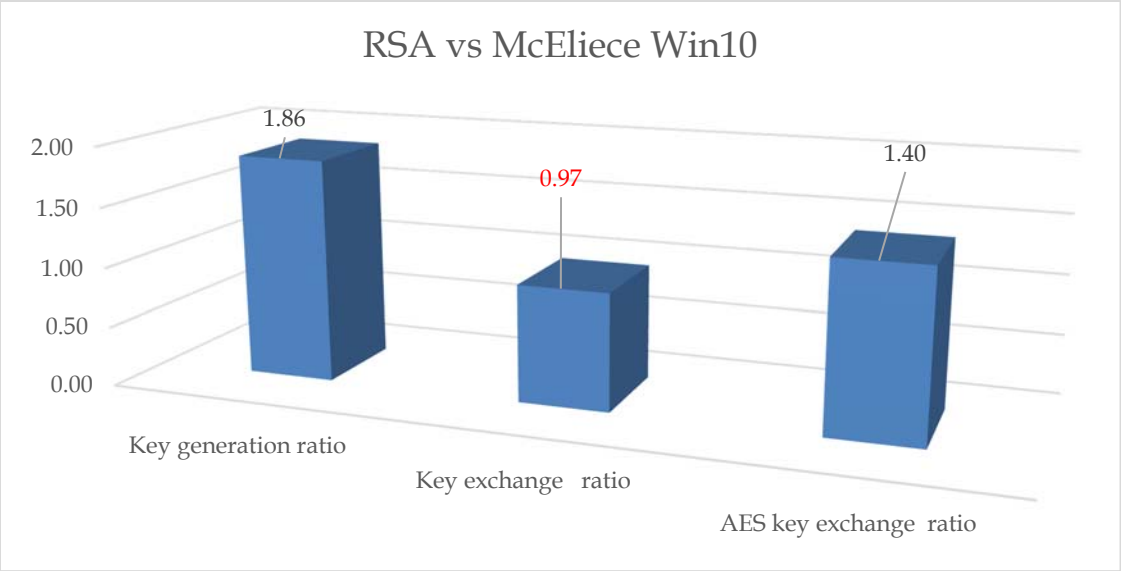
8.5.3 Σύγκριση υλοποίησης RSA vs McEliece στο σύστημα Windows 7



Σχήμα 8.3: $\frac{t_{McEliece}}{t_{RSA}}$ συγκριτικά μόνο στο μηχάνημα που εκτελεί Windows 7 .

Ως προς τις μετρήσεις στο μηχάνημα με windows 7 για την υλοποίηση McEliece-1024 παρατηρούμε ότι χρειάζεται 3 φορές περισσότερος χρόνος για τη δημιουργία του ζεύγους κλειδιών. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζεται σχεδόν ο ίδιος χρόνος. Αυτό οφείλεται τόσο στη διαφορετική υλοποίηση για τον McEliece-1024, ο οποίος έγινε με μεταφορά σε σειριακή μορφή Byte[], αλλά από την εξάρτηση από το μέγεθος κλειδιού και όχι από τον αλγόριθμο. Για την αποστολή του μυστικού κλειδιού χρειάστηκε 2 φορές περισσότερος χρόνος. Εν γένει διαφαίνεται μια διαφοροποίηση προς τα πάνω στον McEliece τόσο στη δημιουργία κλειδιού όσο και στην μεταφορά του μυστικού κλειδιού AES.

8.5.4 Σύγκριση υλοποίησης RSA vs McEliece στο σύστημα Windows 10



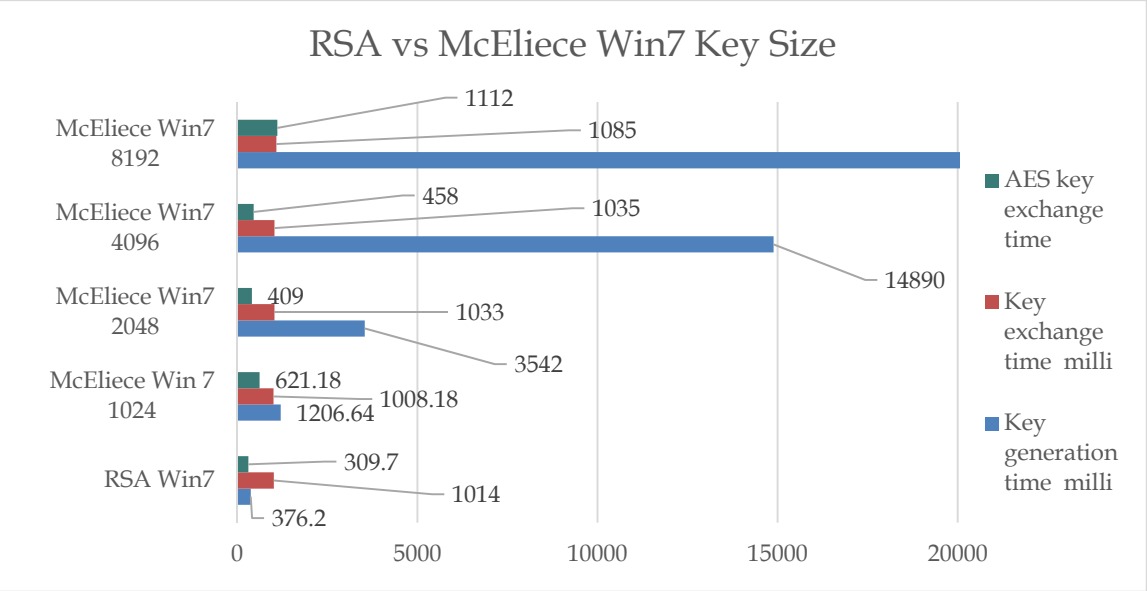
Σχήμα 8.4: $\frac{t_{McEliece}}{t_{RSA}}$ συγκριτικά μόνο στο μηχάνημα που εκτελεί Windows 10

Ως προς τις μετρήσεις στο μηχάνημα με windows 10 για την υλοποίηση McEliece-1024 παρατηρούμε ότι χρειάζεται 2 φορές περισσότερος χρόνος για τη δημιουργία του ζεύγους κλειδιών. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζεται περίπου ο ίδιος χρόνος. Αυτό οφείλεται τόσο στη διαφορετική υλοποίηση για τον McEliece-1024, ο οποίος έγινε με μεταφορά σε σειριακή μορφή Byte[], όσο και από την εξάρτηση από το μέγεθος κλειδιού και όχι από τον αλγόριθμο. Για την αποστολή του μυστικού κλειδιού χρειάστηκε 1,4 φορές περισσότερος χρόνος. Και πάλι διαφαίνεται μια διαφοροποίηση προς τα πάνω τόσο στη δημιουργία κλειδιού όσο και στην μεταφορά του μυστικού κλειδιού AES.

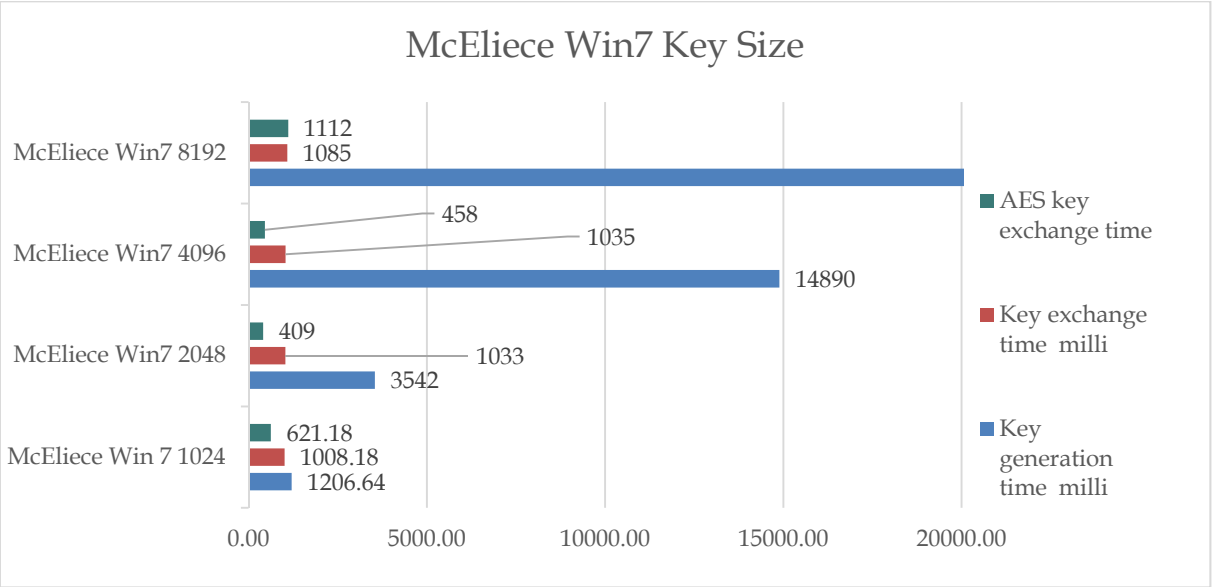
Συγκριτικά βλέπουμε ότι υπάρχει και διαφορά αναλόγως του συστήματος που εκτελούνται οι αλγόριθμοι (κεφάλαια 8.5.3.-8.5.4)

8.5.5 Σύγκριση υλοποίησης RSA vs McEliece μόνο στο σύστημα Windows 7 για διάφορα μεγέθη κλειδιού

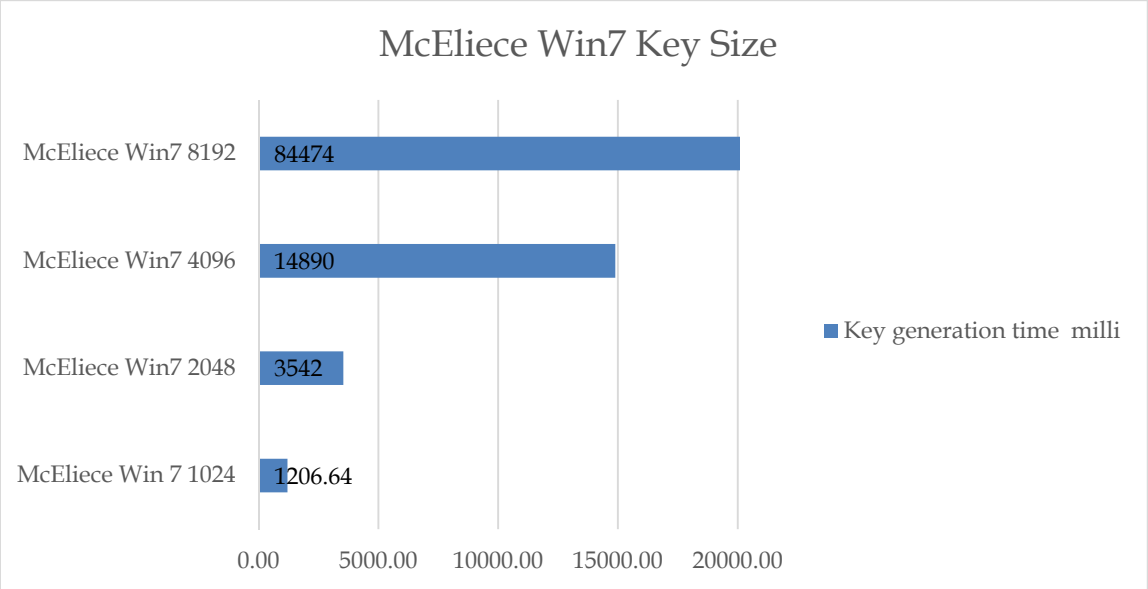
Πιο κάτω κάναμε μια προσέγγιση ώστε να χρησιμοποιηθεί μεγαλύτερο μέγεθος κλειδιού McEliece από το 1024 που χρησιμοποιήθηκε στην υλοποίηση αρχικά. Στο μηχάνημα Windows 7 για μεγέθη κλειδιού 1024, 2048, 4096, 8192, βλέπουμε τα αποτελέσματα και θα ακολουθήσει συνοπτική αποτίμηση.



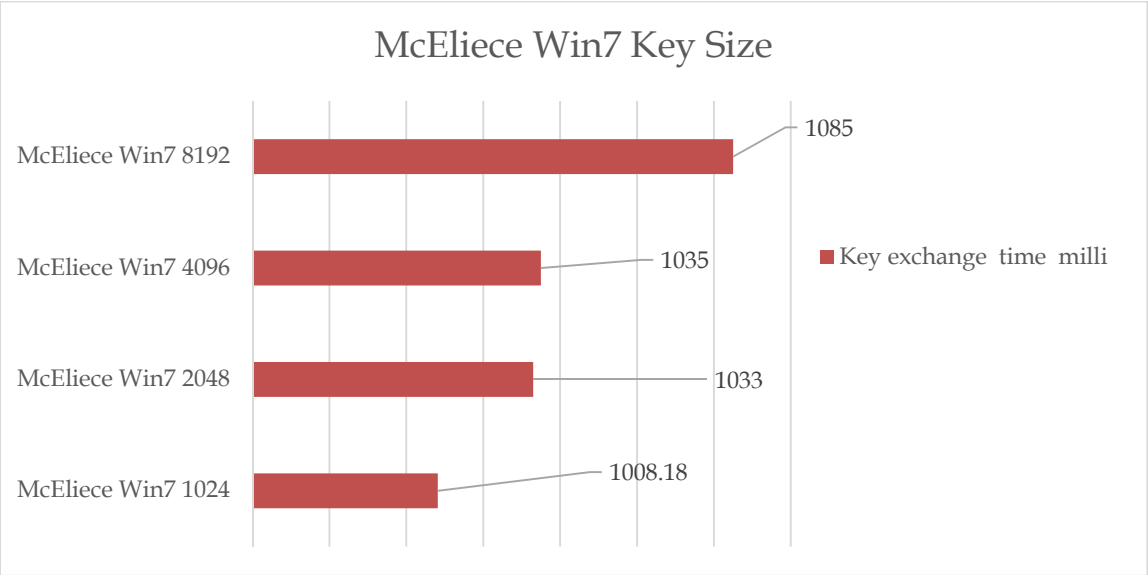
Σχήμα 8.5: Χρόνοι στο σύστημα Windows 7 για τα διαφορά μήκη κλειδιών McEliece και οι αντίστοιχοι χρόνοι για RSA 2048 στο ίδιο σύστημα.



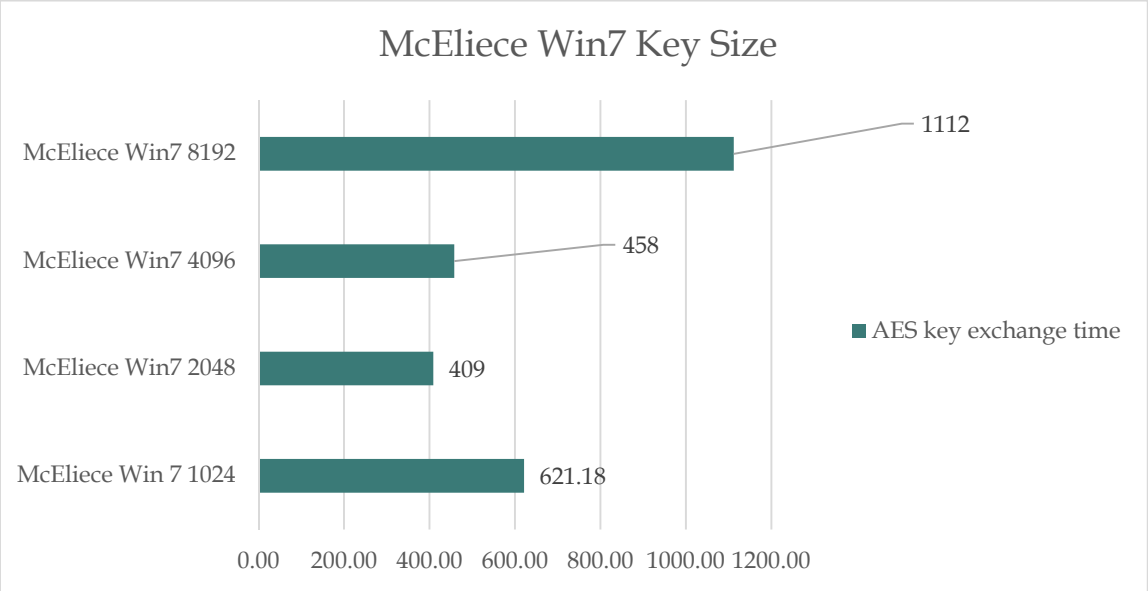
Σχήμα 8.6: Χρόνοι στο σύστημα Windows 7 για τα διάφορα μήκη κλειδιών McEliece



Σχήμα 8.7: Χρόνοι παραγωγής κλειδιού στο σύστημα Windows 7 για τα διάφορα μήκη κλειδιών McEliece



Σχήμα 8.8: Χρόνοι ανταλλαγής δημόσιου κλειδιού McEliece στο σύστημα Windows 7 για τα διάφορα μήκη κλειδιών McEliece.



Σχήμα 8.9: Χρόνοι ανταλλαγής συμμετρικού κλειδιού AES στο σύστημα Windows 7 για τα διάφορα μήκη κλειδιών McEliece.

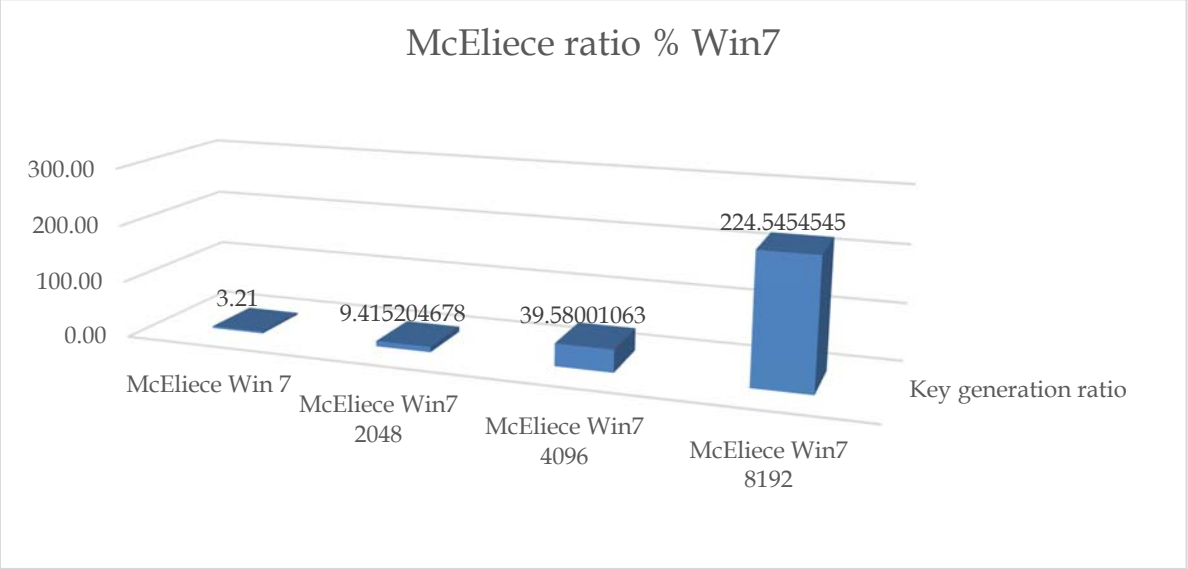
Ως προς τις μετρήσεις των χρόνων για την υλοποίηση McEliece-2048 παρατηρούμε ότι χρειάζονται 3542 ms για τη δημιουργία του ζεύγους κλειδιών. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζονται 1033 ms. Όπως προ είπαμε, η υλοποίηση για τον McEliece-έγινε με μεταφορά σε σειριακή μορφή Byte[]. Για την αποστολή του μυστικού κλειδιού χρειάστηκαν 409 ms.

Ως προς τις μετρήσεις των χρόνων για την υλοποίηση McEliece-4096 παρατηρούμε ότι χρειάζονται 14890 ms για τη δημιουργία του ζεύγους κλειδιών. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζονται 1035 ms. (με μεταφορά σε σειριακή μορφή Byte[]). Για την αποστολή του μυστικού κλειδιού χρειάστηκαν 458 ms.

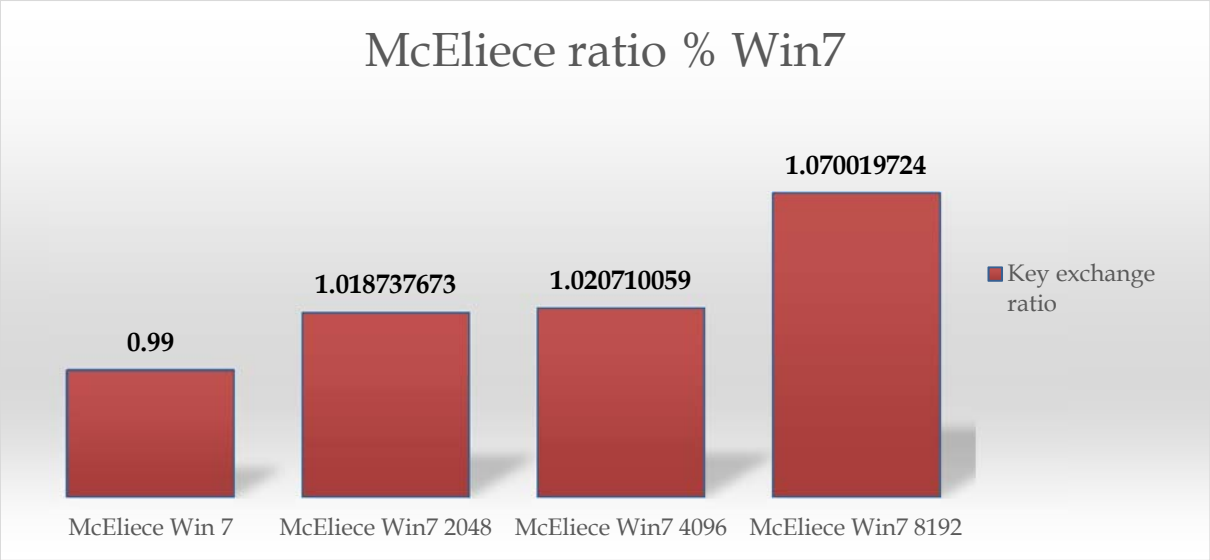
Ως προς τις μετρήσεις των χρόνων για την υλοποίηση McEliece-8192 παρατηρούμε ότι χρειάζονται 84474 ms για τη δημιουργία του ζεύγους κλειδιών. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζονται 1085 ms. (με μεταφορά σε σειριακή μορφή Byte[]). Για την αποστολή του μυστικού κλειδιού χρειάστηκαν 1112 ms.

McEliece -bit	Δημόσιο κλειδί (kB)	Ιδιωτικό κλειδί (kB)
1024	65	104
2048	257	404
4096	1029	1589
8192	4098	6303

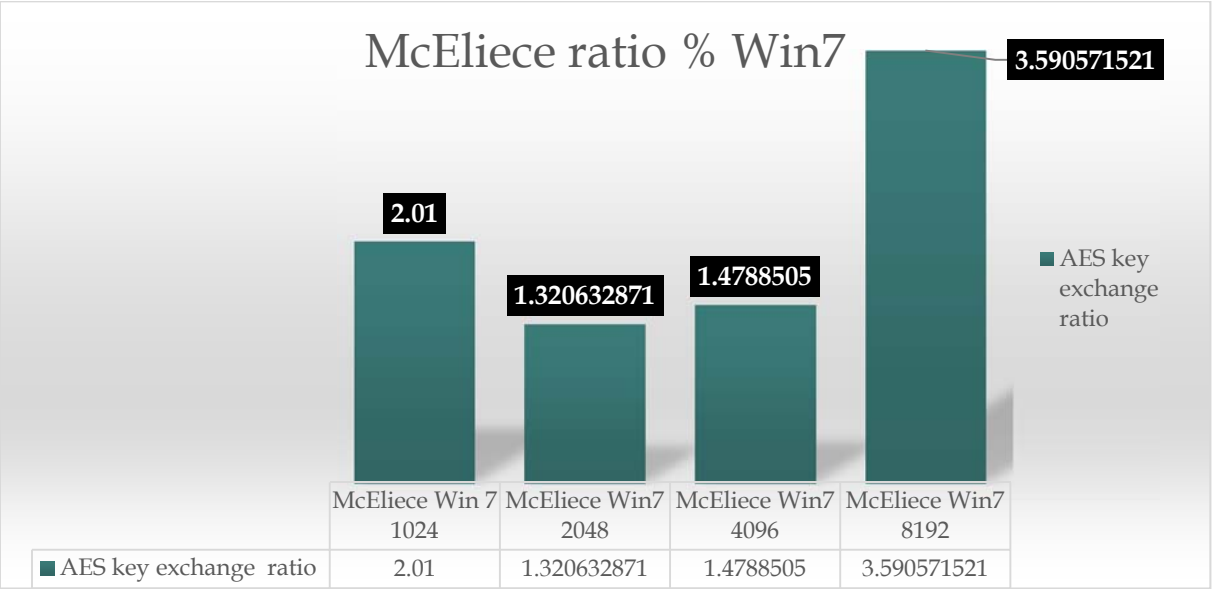
Πίνακας 8.1: Μεγέθη των αρχείων ιδιωτικού και δημόσιου κλειδιού για διαφορά μεγέθη κλειδιού McEliece.



Σχήμα 8.10: $\frac{t_{McEliece}}{t_{RSA}}$ συγκριτικά με τον RSA στο μηχάνημα που εκτελεί Windows 7 για την παραγωγή κλειδιού μόνο.



Σχήμα 8.11: $\frac{t_{McEliece}}{t_{RSA}}$ συγκριτικά με τον RSA στο μηχάνημα που εκτελεί Windows 7 για την ανταλλαγή δημοσίου κλειδιού μόνο.



Σχήμα 8.12: $\frac{t_{McEliece}}{t_{RSA}}$ συγκριτικά με τον RSA στο μηχάνημα που εκτελεί Windows 7 για την ανταλλαγή ιδιωτικού κλειδιού μόνο.

Ως προς τις μετρήσεις των $\frac{t_{McEliece}}{t_{RSA}}$στο μηχάνημα με windows 7 για την υλοποίηση McEliece-1024 παρατηρούμε ότι χρειάζεται 3 φορές περισσότερος χρόνος για τη δημιουργία του ζεύγους κλειδιών. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζεται σχεδόν ο ίδιος χρόνος για τους λόγους που προαναφέρθηκαν. Για την αποστολή του μυστικού κλειδιού χρειάστηκε 2 φορές περισσότερος χρόνος. Εν γένει διαφαίνεται μια διαφοροποίηση προς τα πάνω τόσο στη δημιουργία κλειδιού όσο και στην μεταφορά του μυστικού κλειδιού AES.

Ως προς τις μετρήσεις των $\frac{t_{McEliece}}{t_{RSA}}$στο μηχάνημα με windows 7 για την υλοποίηση McEliece-2048 παρατηρούμε ότι χρειάζεται 9 φορές περισσότερος χρόνος για τη δημιουργία του ζεύγους κλειδιών. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζεται ο ίδιος χρόνος (μεταφορά σε σειριακή μορφή Byte[]). Για την αποστολή του μυστικού κλειδιού χρειάστηκε 1,32 φορές περισσότερος χρόνος. Κι εδώ φαίνεται μια διαφοροποίηση προς τα πάνω τόσο στη δημιουργία κλειδιού όσο και στην μεταφορά του μυστικού κλειδιού AES ενώ η απότομη αύξηση του χρόνου για τη δημιουργία ζεύγους κλειδιών είναι καταφανής.

Ως προς τις μετρήσεις των $\frac{t_{McEliece}}{t_{RSA}}$στο μηχάνημα με windows 7 για την υλοποίηση McEliece-4096 παρατηρούμε ότι χρειάζεται 39 φορές περισσότερος χρόνος για τη δημιουργία του ζεύγους κλειδιών. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζεται σχεδόν ο ίδιος χρόνος. Για την αποστολή του μυστικού κλειδιού χρειάστηκε 1,47 φορές περισσότερος χρόνος. Η αύξηση του χρόνου για τη δημιουργία ζεύγους κλειδιών είναι δραματική.

Ως προς τις μετρήσεις των $\frac{t_{McEliece}}{t_{RSA}}$ στο μηχάνημα με windows 7 για την υλοποίηση McEliece-8192 παρατηρούμε ότι χρειάζεται 224 φορές περισσότερος χρόνος για τη δημιουργία του ζεύγους κλειδιών. Για την ανταλλαγή του δημοσίου κλειδιού μέσω socket χρειάζεται σχεδόν ο ίδιος χρόνος (μεταφορά σε σειριακή μορφή Byte[]) και ήδη παρατηρείται η καθυστέρηση αναλόγως του μεγέθους κλειδιού όπως αναφέρθηκε. Για την αποστολή του μυστικού κλειδιού χρειάστηκε τριπλάσιος χρόνος.

Κεφάλαιο 9
Σύνοψη - μελλοντική έρευνα

Οι προσπάθειες για την εύρεση του κατάλληλου αλγόριθμου εντείνονται καθημερινά και ο συναγωνισμός στο πρόγραμμα PQC του NIST είναι αρκετά εκτεταμένος. Στην παρούσα εργασία, μελετήσαμε σε πειραματικό επίπεδο μια εφαρμογή συνομιλίας με κρυπτογράφηση δημοσίου αλλά και συμμετρικού κλειδιού όπως συνηθίζεται στις εφαρμογές του Διαδικτύου.

Από τις μετρήσεις και την ανάλυσή μας έχουμε καταλήξει σε κάποια ευρήματα. Συμπερασματικά βλέπουμε ότι ο χρόνος δημιουργίας κλειδιού στο McEliece είναι προφανώς μεγαλύτερος και όσο αυξάνεται το μέγεθος κλειδιού μεγαλώνει ακόμα περισσότερο αφού για παράδειγμα με μήκος κλειδιού 4 MByte έχουμε βρει μέσω της υλοποίησης μας ότι χρειάζεται 80 (αναλόγως συστήματος) δευτερόλεπτα για τη δημιουργία του ζεύγους.

Ως προς την αποστολή και ανταλλαγή δημόσιου κλειδιού βλέπουμε ότι οι χρόνοι είναι σχετικά ίδιοι. Μπορούμε να βασιστούμε για αυτό στη γνώση ότι η μεταφορά ενός κλειδιού σαν αντικείμενο σε java το οποίο και εφαρμόστηκε για τον αλγόριθμο RSA, παίρνει περισσότερο χρόνο από την

από την μεταφορά σε δυαδικό πίνακα που εφαρμόσαμε στην περίπτωση του αλγορίθμου McEliece.Ο χρόνος αυτός αφορά στην υλοποίηση και όχι στον αλγόριθμο οπότε αυξάνει αρκετά μόνο στην περίπτωση της υλοποίησης McEliece μεγάλου μεγέθους κλειδιού. Επίσης στην υλοποίηση McEliece-1024 δεν υπεισέρχεται μεγάλο μέγεθος κλειδιού και είναι της τάξης των 65 kB οπότε δεν περιμένουμε να πάρει περισσότερο χρόνο αφού και τα δύο μέρη εκτελούνται στο τοπικό μηχάνημα. Για την μεταφορά του δημόσιου κλειδιού παρατηρούνται χρόνοι γύρω στο δευτερόλεπτο. Στην ουσία όμως αυτό θα μπορούσε εύκολα να επηρεαστεί από μικρό εύρος καναλιού (στο Διαδίκτυο) γιατί τα κλειδιά είναι πάνω από 1MB σ τον McEliece-4096 για παράδειγμα. Το ίδιο ισχύει και για τη μεταφορά του συμμετρικού κλειδιού ως προς το χρόνο μεταφοράς σε κανάλι όχι ευρείας ζώνης (narrow bandwidth)

Ως προς την αποστολή ιδιωτικού κλειδιού βλέπουμε ότι οι χρόνοι αυξάνονται κατά 2-3 φορές στο κρυπτοσυστημα McEliece. Αυτό φαίνεται λογικό και δείχνει μια σωστή τάση αφού η πολυπλοκότητα στους υπολογισμούς στον RSA είναι μικρότερη έναντι του McEliece. Εν γένει ο αλγόριθμος δημοσίου κλειδιού McEliece είναι αρκετά κοστοβόρος και χρονοβόρος πάντα βάσει της υλοποίησής μας. Αυτό βέβαια συνάδει και με την σχετική πολυπλοκότητα των αλγορίθμων αυτών.

Επιπλέον παρατηρούμε ότι θα συνεχίσει να διαδραματίζει σημαντικό ρόλο η πλατφόρμα στην οποία θα υλοποιείται ένας αλγόριθμος και στην εποχή των κβαντικών υπολογιστών. Είναι προφανές ότι θα υπάρχουν διάφορες υλοποιήσεις λογισμικού αλλά και υλισμικού. Οπότε πάντα θα αναμένονται διαφορές στις εκτελέσεις ίδιου πρωτοκόλλου σε διαφορετικά μηχανήματα όπως συμβαίνει πάντα στους υπολογιστές.

Στο μέλλον θα περιμένουμε την έγκριση ενός αλγόριθμου ή σχήματος τόσο για την κρυπτογραφία δημόσιου κλειδιού όσο και για τις ψηφιακές υπογραφές. Εντωμεταξύ, θα πρέπει ήδη η επιστημονική κοινότητα αλλά και οι εταιρείες που παρέχουν ασφαλείς λύσεις στον τελικό χρήστη να δουν με σκεπτικισμό την μετακβαντική εποχή. Καθίσταται απαραίτητη η διαφύλαξη κρυπτογραφημένων δεδομένων τα οποία υπάρχουν σήμερα επιπροσθέτως με χρήση μετακβαντικής κρυπτογραφίας. Εν ολίγοις, θα ήταν πολύ επίφοβη η επιλογή της αμιγούς εφαρμογής μετακβαντικών αλγορίθμων στο μέλλον. Ας μην ξεχνούμε πόσο μεγάλος όγκος κρυπτογραφημένων δεδομένων υπάρχει, με αλγόριθμους που δεν αντέχουν την επίθεση μετακβαντικά. Θα ήταν πολύ άβολο, τα δεδομένα αυτά να γίνουν έρμαιο brute-force στη μετακβαντική εποχή. Προφανώς, θα εκτεθούν πολύ εύκολα και η ιδιωτικότητα τους δε θα δύναται να διαφυλαχθεί.

Τέλος, είναι σημαντική η έγκαιρη και έγκυρη ανεύρεση ενός μοντέλου μετάβασης από την προ κβαντική στην μετακβαντική εποχή. Δεν είναι δυνατόν μέσα σε λίγο καιρό από την λήψη της απόφασης υλοποίησης του μετακβαντικού πρωτοκόλλου, να ενημερωθούν ευρέως καθιερωμένες εφαρμογές. Αυτό θα πάρει αρκετό χρόνο. Κατά την άποψή μας θα ήταν σωστότερη μια διττή αντιμετώπιση του ζητήματος. Δηλαδή η σχεδίαση και η ανάπτυξη υπαρχόντων εφαρμογών και δομών να αρχίσει να συμπεριλαμβάνει στην αρχιτεκτονική του κώδικα, περεταίρω δυνατότητες κρυπτογράφησης εκτός από τις ήδη διαδεδομένες. Για παράδειγμα, μια πλατφόρμα συνομιλίας που χρησιμοποιεί PKCS, να δώσει τη δυνατότητα χρήσης και άλλου μετακβαντικού πρωτοκόλλου. Ακόμα και την πειραματική υλοποίηση μετακβαντικού αλγοριθμου προ της προτυποποίησης από το NIST. Έτσι, μετά την έλευση του πρωτοκόλλου από το NIST και την προτυποποίηση του , θα μπορεί η εφαρμογή να δουλεύει και κλασσικά αλλά και μετακβαντικά ταυτόχρονα μέχρι η μετακβαντική τεχνολογία να επικρατήσει. Αυτό το θα μπορούσαμε να το αντιπαραβάλουμε με τη χρήση Dual Stack δρομολογητών IPv4 IPv6 κατά την αντίστοιχη μετάβαση από το ένα σχήμα στο άλλο.

Βιβλιογραφία

[01] E. Alkim, L. Ducas, T. Pöppelmann, P. Schwabe. "Post-quantum key exchange – a new hope", 2015.

[02] S. Au, C. Eubanks-Turner, J. Everson, "The McEliece Cryptosystem", 2003.

[03] D. Bernstein, "Grover vs. McEliece", Chicago: The University of Illinois at Chicago, 2009.

[04] D. Bernstein, J. Buchmann,E. Dahmen, "Post-Quantum Cryptography", Springer, 2009.

[05] D. Bernstein, T. Chou, P. Schwabe, “McBits:Fast constant-time code-based cryptography”, 2016.

[06] D. Bernstein, T. Lange. "Post-quantum cryptography –dealing with the fallout of physics success", 2009.

[07] R. Biswas, S. Bandyopadhyay, A. Banerjee. "A Fast implementation of the RSA algorithm using the GNU MP library", Calcutta: IIIT-Calcutta

[08] D. Boneh, M. Franklin. "Identity-based encryption from the weil pairing", SIAM J. of Computing, Vol. 32 No. 3 (2003): 586-615.

[09] J. Bos, M. Kaihara, T. Kleinjung, A. Lenstra, P. Montgomery, "On the Security of 1024-bit RSA", 2009.

[10] V. Gauthier, R. Knudsen, G. Leander, "Post-Quantum Cryptography", Technical University of Denmark, 2011.

[11] L. Grover, "A fast quantum mechanical algorithm for database search", Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing 1996.

[12] E. Jochemsz, "Goppa Codes & the McEliece Cryptosystem", Amsterdam: Vrije Universiteit, 2002.

[13] P. Kaye, R. Laflamme, M. Mosca, "An Introduction to Quantum Computing", New York: Oxford University Press Inc, 2007.

[14] M. Mariantoni, "Building a Superconducting Quantum Computer", 2016.

[15] R.J McEliece, "A public-key cryptosystem based on algebraic coding theory", DSN Progress Report February 1978.

[16] A. Menezes, P. van Oorschot, S. Vanstone, "Handbook of applied Cryptography", 1996.

[17] R. Niebuhr, M. Meziani, S. Bulygin, J. Buchmann, "Selecting Parameters for Secure McEliece-based Cryptosystems", 2010.

[18] R. Niederhagen, M. Waidner, "Practical Post-Quantum Cryptography, 2017.

[19] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory. Problems of Control and Information Theory" (pp. 159-166), 1986.

[20] J. Nikas, "How Google's Quantum Computer Could Change the World", 16 October 2017, WSJ, <www.wsj.com>.

[21] NIST, "Post-Quantum-Cryptography, Round-1-Submissions", 2016, <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions>.

[22] NIST, "Report on Post-Quantum Cryptography", 2016.

[23] C. Shannon, "A Mathematical Theory of Communication", Bell System Technical J ournal, July, 1948.

[24] T. Simonite, "Moore's Law Is Dead. Now What?" 13 March 2016, <https://www.technologyreview.com/s/601441/moores-law-is-dead-now-what/>.

[25] W. Stallings, «Κρυπτογραφία και ασφάλεια δικτύων», Εκδοτικός Όμιλος Ίων, 2012.

[26] T. Stockinger, "GSM network and its privacy - the A5 stream cipher", 2005.

[27] G. Tsudik, "Secure Computing & Network Center, Computer & Network Security <http://sconce.ics.uci.edu/134-W14>.

[28] V. Umana, phD Thesis, " Post-Quantum Cryptography",2011.

[29] M. Wiener, "Cryptanalysis of short RSA secret exponents", 1990.

[30] Wikipedia, <https://commons.wikimedia.org>.

[31] Δ. ΒΑΡΣΟΣ, «Μια εισαγωγή στην αλγεβρική θεωρία κωδίκων», Αθήνα: ΣΕΑΒ, 2015.

[32] Σ. Γκρίτζαλης, Σ. Κάτσικας, Β. Χρυσικόπουλος, Μ. Burmester, «Σύγχρονη κρυπτογραφία, Θεωρία και εφαρμογές». Παπασωτηρίου, n.d.

[33] Ε. Ζάχος, Α. Παγουρτζής, Π. Γροντάς, «Υπολογιστική Κρυπτογραφία», Αθήνα: ΣΕΑΒ, 2015.

[34] Β. Κάτος, Γ. Στεφανίδης, «Τεχνικές Κρυπτογραφίας και Κρυπτανάλυσης», εκδόσεις ΖΥΓΟΣ,2003.

Παράρτημα Α

Πηγαίος κώδικας

Α.1 Πηγαίος κώδικας για το 8.3

Client Class:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.nio.file.Files;
import java.security.KeyPair;
import java.security.NoSuchAlgorithmException;
```

```
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.Calendar;

import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;


public class ClientChat extends JFrame implements ActionListener {
  static Socket con;
  JPanel panel;
  JTextField NewMsg;
  JTextArea ChatHistory;
  JButton Send;
  String line;
                BufferedReader br;
                String response;
                BufferedReader is ;
                PrintWriter os;

 PublicKey ServerpublicKey;
static PublicKey Clientpublickey ;
static PrivateKey Clientprivatekey;
static String IP="127.0.0.1";
//static String IP="192.168.1.7";

  public ClientChat() throws Exception {
    panel = new JPanel();
    NewMsg = new JTextField();
    ChatHistory = new JTextArea();
    Send = new JButton("send");
    this.setSize(500, 500);
    this.setVisible(true);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    panel.setLayout(null);
    this.add(panel);
    ChatHistory.setBounds(20, 20, 450, 360);
    panel.add(ChatHistory);
    NewMsg.setBounds(20, 400, 340, 30);
    panel.add(NewMsg);
    Send.setBounds(375, 400, 95, 30);
    panel.add(Send);
    Send.addActionListener(this);
    con = new Socket(InetAddress.getByName(IP),2000);
    ChatHistory.setText("Connected to Server");
    GeTtimestamp("RSA Keypair creation START");
    long startTime = System.nanoTime();
    //Generate local RSA keys public private
    KeyPair keypair = createkeypair();
                                PublicKey Clientpublickey = keypair.getPublic();
                                PrivateKey Clientprivatekey = keypair.getPrivate();
                                 long endTime = System.nanoTime();
                                 long duration = (endTime - startTime)/1000000 ;
                                 GeTtimestamp("RSA Keypair creation END  "+duration+"  milliseconds");
                                //Send public key to Server
                                 WriteKeystoFile(keypair);
                                 GeTtimestamp("RSA Public key exchange START");
                                 long startTime1 = System.nanoTime();
                                sendclientpubKey(keypair.getPublic());
                                //ChatHistory.setText(ChatHistory.getText() + '\n' + "Sent public key to SERVER ");
    //Get servers public key
                                PublicKey ServerpublicKey = receiveserverpubKey();
                                //ChatHistory.setText(ChatHistory.getText() + '\n' + "Received SERVER public key ");
                                //And save to file
                                //X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(ServerpublicKey.getEncoded());
                                 long endTime1 = System.nanoTime();
                                 long duration1 = (endTime1 - startTime1)/1000000 ;
                                 GeTtimestamp("RSA Public key exchange END  "+duration1+"  milliseconds");
                                FileOutputStream fos = new FileOutputStream("/home/antony/Client/Serverpublic.key");
                                fos.write(ServerpublicKey.getEncoded());
                                fos.close();
                                 GeTtimestamp("RSA AES key exchange START");
```

```
                    long startTime2 = System.nanoTime();
                    receiveSecret(Clientprivatekey );
                    long endTime2 = System.nanoTime();
                    long duration2 = ((endTime2 - startTime2)/1000000 );
                    GeTtimestamp("RSA AES key exchange END  "+duration2+"  milliseconds");

 this.setTitle("Secure Client");
 ChatHistory.setText(ChatHistory.getText() + '\n' + "RSA chat Secured on APP Level started ");
 while (true) {
  try {
          //Read and recreate AES from file
          SecretKey secret = getAESkeyfromfile();
          //System.out.println("CLIENT MAIN: Key is "+ secret) ;
          DataInputStream dis = new DataInputStream(con.getInputStream());
          //System.out.println("CLIENT MAIN: DataInputStream OPENED") ;
          int length = dis.readInt();
    byte[] cyphertextbytes = new byte[length];
    dis.readFully(cyphertextbytes,0,cyphertextbytes.length);
    //System.out.println("CLIENT MAIN: Read something") ;
    //System.out.println("CLIENT MAIN: encrypted in bytes:" + cyphertextbytes) ;
    //byte[] cyphertextbytes = cyphertext.getBytes();
    String decryptedcyphertext =AES.decryptText(secret, cyphertextbytes) ;
    //System.out.println("CLIENT MAIN: decrypted :" + decryptedcyphertext) ;

    //simple implementation String line = is.readLine();
    ChatHistory.setText(ChatHistory.getText() + '\n' + "Server:"
      +new String(decryptedcyphertext));

  } catch (Exception e1) {
    ChatHistory.setText(ChatHistory.getText() +'\n'
      + "Message sending fail:Network Error");
    try {
     Thread.sleep(3000);
     System.exit(0);
    } catch (InterruptedException e) {
     // TODO Auto-generated catch block
     e.printStackTrace();
    }
  }
 }
}

@Override
public void actionPerformed(ActionEvent e) {
 // TODO Auto-generated method stub

 if ((e.getSource() == Send) && (NewMsg.getText() != "")) {

   ChatHistory.setText(ChatHistory.getText() + '\n' + "Me:"
     + NewMsg.getText());
   try
   {

   SecretKey secret = getAESkeyfromfile();
           DataOutputStream dos = new DataOutputStream(
                                                   con.getOutputStream());
                                   String text =NewMsg.getText();
                                   byte[] textbytes =                text.getBytes();
                                   byte[] cyphertextbytes=AES.encryptText(secret, text);
                                   dos.writeInt(cyphertextbytes.length);
                                   dos.write(cyphertextbytes);
                                   //System.out.println( "CLIENT MAIN:Text to SEND is " +NewMsg.getText());
                                   //System.out.println( "CLIENT MAIN: Encrypted Text bytes to SEND is " +cyphertextbytes);
                                   //dos.flush();

   } catch (Exception e1) {
    ChatHistory.append(ChatHistory.getText() + '\n'
      + "Message sending fail:Network Error");

   }


   NewMsg.setText("");
  }
}

public static KeyPair createkeypair() throws NoSuchAlgorithmException{
```

```
                KeyPair keypair = RSA.buildKeyPair();
                PublicKey ClientpublicKey = keypair.getPublic();
                PrivateKey ClientprivateKey = keypair.getPrivate();
                return keypair;
        }

public void  WriteKeystoFile(KeyPair keypair) throws IOException {
        PublicKey ClientpublicKey = keypair.getPublic();
        PrivateKey ClientprivateKey = keypair.getPrivate();
                FileOutputStream fosPub = new FileOutputStream("/home/antony/Client/Clientpublic.key");
                fosPub.write(ClientpublicKey.getEncoded());
                fosPub.close();
                FileOutputStream fosPriv = new FileOutputStream("/home/antony/Client/Clientprivate.key");
                fosPriv.write(ClientprivateKey.getEncoded());
                fosPriv.close();
        }



public static SecretKey getAESkeyfromfile() throws IOException {
        byte []keybyte = new byte[32];
        FileInputStream fin = new FileInputStream("/home/antony/Client/secret.key");
        fin.read(keybyte);
        SecretKey secret = new SecretKeySpec(keybyte,"AES");
        //System.out.println("CLIENT SECRET IS:" + secret );
        fin.close();
        return secret;
}
public static PublicKey sendclientpubKey(PublicKey ClientpublicKey) throws IOException, ClassNotFoundException{
        Socket clientSocket = new Socket(IP,1979);
        clientSocket.setTcpNoDelay(true);
                //System.out.println("Client Socket 2 opened");
                ObjectOutputStream Clientpubkeystream = new ObjectOutputStream(clientSocket.getOutputStream());
                try {Clientpubkeystream.writeObject(ClientpublicKey);
                //System.out.println("Client Key SENT "+ClientpublicKey);
                } catch (Exception e1)
                                        {System.out.println("Client Key sending failure:Network issues");
                                        }
                Clientpubkeystream.flush();
                Clientpubkeystream.close();
                clientSocket.close();
                //System.out.println("Client Socket 2 closed");
                return ClientpublicKey;
}



public static PublicKey receiveserverpubKey() throws IOException, ClassNotFoundException{
        Socket clientSocket = new Socket(IP,1978);
        clientSocket.setTcpNoDelay(true);
                //System.out.println("Client Socket 1 opened");
                ObjectInputStream Serverpubkeystream = new ObjectInputStream(clientSocket.getInputStream());
                PublicKey ServerpublicKey = (PublicKey) Serverpubkeystream.readObject();
                //System.out.println("Server Key RECEIVED "+ServerpublicKey);

                Serverpubkeystream.close();
                clientSocket.close();

                 //System.out.println("Client Socket 1 closed");
                 return ServerpublicKey;
}


public static void receiveSecret(PrivateKey myRSAprivate ) throws Exception{

        Socket clientSocket = new Socket(IP,1980);
        clientSocket.setTcpNoDelay(true);
        //System.out.println("Client Socket 3 opened");
        byte [] mybytearray =new byte[256];
          InputStream is = clientSocket.getInputStream();
          FileOutputStream fos = new FileOutputStream("/home/antony/Client/ENCsecret.key");
          BufferedOutputStream bos = new BufferedOutputStream(fos);
          int bytesRead = is.read(mybytearray,0,mybytearray.length);
          int current = bytesRead;
          do {
            bytesRead =
             is.read(mybytearray, current, (mybytearray.length-current));
```

```
                if(bytesRead >= 0) current += bytesRead;
               } while(bytesRead<-1);

               bos.write(mybytearray, 0 , current);
               bos.flush();

               if (fos != null) fos.close();
               if (bos != null) bos.close();
               if (clientSocket != null) clientSocket.close();

              //System.out.println("File " + "/home/antony/Client/ENCsecret.key"+ " downloaded " + mybytearray.length + " bytes
read)");
               File file = new File("/home/antony/Client/ENCsecret.key");
               //System.out.println("File " +"/home/antony/Client/ENCsecret.key"+ file.length()+ " bytes ");
                        //System.out.println("RECEIVED Secret key is"+mybytearray);
             //System.out.println("Client Socket 3 closed");

             //convert encrypted AES file to bytes
             byte[] EncrAESbyte = Files.readAllBytes(new File("/home/antony/Client/ENCsecret.key").toPath());
             //System.out.println("encrypted AES byte array is :" +EncrAESbyte.length+"bytes");
             SecretKey AESkey =RSA.decryptAES(myRSAprivate, EncrAESbyte);
             byte[] AESbyte=AESkey.getEncoded();
             //Write AES to file
             FileOutputStream ffos = new FileOutputStream("/home/antony/Client/secret.key");
             ffos.write(AESbyte, 0, AESbyte.length);
             ffos.close();
             //Get back AES key from file
             byte[] AESbytefromfile = Files.readAllBytes(new File("/home/antony/Client/secret.key").toPath());
             SecretKeySpec secretKeySpec = new SecretKeySpec(AESbytefromfile, "AES");

  }
 public static void GeTtimestamp(String e) {
             Calendar calendar = Calendar.getInstance();
             java.util.Date currentTimestamp = new java.sql.Timestamp(calendar.getTime().getTime());
  System.out.println(currentTimestamp+e);

             }


 public static void main(String[] args) throws Exception {
   new ClientChat () ;
 }
}
```

Server Class:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.KeyPair;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
```

```
import javax.swing.JTextField;

public class ServerChat extends JFrame implements ActionListener {
  static ServerSocket server;
  static Socket conn;
  JPanel panel;
  JTextField NewMsg;
  JTextArea ChatHistory;
  JButton Send;

                static PrivateKey Serverprivate;
                static PublicKey ClientpublicKey1;
                static String IP="127.0.0.1";
                //static String IP="192.168.1.7";
   String line;
   BufferedReader is ;
   public ServerChatForm1() throws Exception {

   panel = new JPanel();
   NewMsg = new JTextField();
   ChatHistory = new JTextArea();
   Send = new JButton("send");
   this.setSize(500, 500);
   this.setVisible(true);
   setDefaultCloseOperation(EXIT_ON_CLOSE);
   panel.setLayout(null);
   this.add(panel);
   ChatHistory.setBounds(20, 20, 450, 360);
   panel.add(ChatHistory);
   NewMsg.setBounds(20, 400, 340, 30);
   panel.add(NewMsg);
   Send.setBounds(375, 400, 95, 30);
   panel.add(Send);
   this.setTitle("Secure Server");
   Send.addActionListener(this);
   //server = new ServerSocket(2000, 1, InetAddress.getLocalHost());
   server = new ServerSocket(2000, 1, InetAddress.getByName(IP));
   ChatHistory.setText("Waiting for Client");
   conn = server.accept();

   //Create AES 256 symmetric key and store it to file
   SecretKey secret = AES.buildKey();
   WriteSecrettoFile(secret);


   //Antony Generate local RSA keys public private

                        KeyPair keypair = createkeypair();
                        //PrivateKey Serverprivate = keypair.getPrivate();
                        //System.out.println("SERVER MAIN: createkeypair finished");
                        //ChatHistory.setText(ChatHistory.getText() + '\n' + "creation of keypair finished");


  //AntonyGet Clients public key

                        PublicKey ClientpublicKey = receiveclientpubKey();
                        FileOutputStream fos = new FileOutputStream("/home/antony/Server/Clientpublic.key");
                        fos.write(ClientpublicKey.getEncoded());
                        fos.close();
                        //ChatHistory.setText(ChatHistory.getText() + '\n' + "Received Client PUBLIC KEY");
  //AntonySend public key to Client
                        sendserverpubKey(keypair.getPublic());
                        //ChatHistory.setText(ChatHistory.getText() + '\n' + "Sent public key to Client ");


             //Send AES to Client
                        sendSecretKey(secret,ClientpublicKey);


   ChatHistory.setText(ChatHistory.getText() + '\n' + "Client Found");
   ChatHistory.setText(ChatHistory.getText() + '\n' + "RSA chat Secured on APP Level started ");
   while (true) {
    try {
             //Read and recreate AES from file
             SecretKey secret1 = getAESkeyfromfile();
              //System.out.println("SERVER MAIN: Secret key from file is:" + secret1);
             DataInputStream dis = new DataInputStream(conn.getInputStream());
```

```
            //System.out.println("SERVER MAIN: DataInputStream OPENED") ;
            int length = dis.readInt();
    byte[] cyphertextbytes = new byte[length];
    dis.readFully(cyphertextbytes,0,cyphertextbytes.length);
    //System.out.println("SERVER MAIN: encrypted in bytes:" + cyphertextbytes) ;
    String decryptedcyphertext =AES.decryptText(secret1, cyphertextbytes) ;
    //System.out.println("SERVER MAIN: Decrypted in bytes:" + decryptedcyphertext) ;

    ChatHistory.setText(ChatHistory.getText() + '\n' + "Server:"
       + new String(decryptedcyphertext));

   } catch (Exception e1)
       {
    ChatHistory.setText(ChatHistory.getText() + '\n'
       + "Message sending fail:Network Error");
    try {
     Thread.sleep(3000);
     System.exit(0);
    } catch (InterruptedException e) {
     // TODO Auto-generated catch block
     e.printStackTrace();
    }
   }
  }
}



@Override
public void actionPerformed(ActionEvent e) {
 // TODO Auto-generated method stub
 if ((e.getSource() == Send) && (NewMsg.getText() != "")) {
  ChatHistory.setText(ChatHistory.getText() + '\n' + "ME:"
     + NewMsg.getText());
  try {
            //Read and recreate AES from file
            SecretKey secret = getAESkeyfromfile();
            //System.out.println("SERVER MAIN: key is :" +secret);

            DataOutputStream dos = new DataOutputStream(
                                                    conn.getOutputStream());
                                            String text =NewMsg.getText();
                                            //byte[] textbytes =text.getBytes();
                                            byte[] cyphertextbytes=AES.encryptText(secret, text);
                                            dos.writeInt(cyphertextbytes.length);
                                            dos.write(cyphertextbytes);
                                            //dos.flush();
                                            //System.out.println("SERVER MAIN: cyphertextbytes was written to socket" ) ;

  } catch (Exception e1) {
    try {
     Thread.sleep(3000);
     System.exit(0);
    } catch (InterruptedException e2) {
     // TODO Auto-generated catch block
     e2.printStackTrace();
    }
  }

  NewMsg.setText("");
 }
}


public static KeyPair createkeypair() throws NoSuchAlgorithmException{
                        KeyPair keyPair = RSA.buildKeyPair();
                        //PublicKey ServerpublicKey = keyPair.getPublic();
                        //PrivateKey ServerprivateKey = keyPair.getPrivate();
                        return keyPair;
            }


public void  WriteSecrettoFile(SecretKey secret ) throws IOException {

                        FileOutputStream fosPub = new FileOutputStream("/home/antony/Server/secret.key");
                        fosPub.write(secret.getEncoded());
```

```
                fosPub.close();

        }

public static SecretKey getAESkeyfromfile() throws IOException {
        byte []keybyte = new byte[32];
        FileInputStream fin = new FileInputStream("/home/antony/Server/secret.key");
        fin.read(keybyte);
        SecretKey secret = new SecretKeySpec(keybyte,"AES");
        fin.close();
        return secret;
}


public static PublicKey sendserverpubKey(Object ServerpublicKey) throws IOException, ClassNotFoundException{
        ServerSocket server = new ServerSocket(1978);
                Socket s=server.accept();
                s.setTcpNoDelay(true);
                //System.out.println("Server Socket 1 opened");
                ObjectOutputStream Serverpubkeystream = new ObjectOutputStream(s.getOutputStream());
                try {Serverpubkeystream.writeObject(ServerpublicKey);
                //System.out.println("Server Key is "+ServerpublicKey);
                //*System.out.println("SERVER Key SENT ");
                } catch (Exception e1)
                {System.out.println("Server Key sending failure:Network issues");
                }
                Serverpubkeystream.flush();
                Serverpubkeystream.close();

                s.close();
                server.close();
                 // System.out.println("Server Socket 1 closed");
                 return null;
}
        //Try to receive  public Client key
public static PublicKey receiveclientpubKey( ) throws IOException, ClassNotFoundException{
        ServerSocket server = new ServerSocket(1979);
                Socket s=server.accept();
                s.setTcpNoDelay(true);
                //System.out.println("Server Socket 2 opened");
                ObjectInputStream clientpubkeystream = new ObjectInputStream(s.getInputStream());
                PublicKey ClientpublicKey1 = (PublicKey) clientpubkeystream.readObject();
                //System.out.println("Client Key as received is "+ClientpublicKey1);
                //*System.out.println("SERVER Key SENT ");
                clientpubkeystream.close();
                s.close();
                server.close();
                //System.out.println("Server Socket 2 closed");
                return ClientpublicKey1 ;
}


public void sendSecretKey(SecretKey secret,PublicKey clientpublic) throws Exception{
        ServerSocket server1 = new ServerSocket(1980, 1, InetAddress.getByName(IP));
        //ServerSocket server = new ServerSocket(1978);

        Socket s=server1.accept();
        s.setTcpNoDelay(true);
                //System.out.println("Server Socket 3 opened");
                byte[] secretbyte = secret.getEncoded();
                byte[] encryptedsecret = RSA.encryptAES(clientpublic, secretbyte);
                File encryptedsecretfile = new File ("/home/antony/Server/ENCsecret.key");
                FileOutputStream fosPub = new FileOutputStream(encryptedsecretfile);
                fosPub.write(encryptedsecret);
                fosPub.close();
                byte [] mybytearray = new byte [(int)encryptedsecretfile.length()];
 FileInputStream fis = new FileInputStream(encryptedsecretfile);
 BufferedInputStream bis = new BufferedInputStream(fis);
 bis.read(mybytearray, 0, mybytearray.length);
 OutputStream os = s.getOutputStream();
 os.write(mybytearray, 0, mybytearray.length);
 //System.out.println("Sending " + "/home/antony/Server/ENCsecret.key" + "(" + mybytearray.length + " bytes)");
 os.flush();
 if (bis != null) bis.close();
 if (os != null) os.close();
 if (s!=null) s.close();
                server1.close();
```

```
                    //System.out.println("Server Secret key is"+mybytearray);
                    //System.out.println("Server Socket 3 closed");

 }
 public static void main(String[] args) throws Exception {
 new ServerChat();
 }}
```

RSA Class:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.Arrays;

import javax.crypto.Cipher;
import java.io.*;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
public class RSA {

        //From object to bytes
        public static byte[] serialize(Object obj) throws IOException {
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        ObjectOutputStream os = new ObjectOutputStream(out);
        os.writeObject(obj);
        return out.toByteArray();
        }//From bytes to object
        public static Object deserialize(byte[] data) throws IOException, ClassNotFoundException {
        ByteArrayInputStream in = new ByteArrayInputStream(data);
        ObjectInputStream is = new ObjectInputStream(in);
        return is.readObject();
        }

 public static KeyPair buildKeyPair() throws NoSuchAlgorithmException
 {
  final int keySize = 2048;
  KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
  keyPairGenerator.initialize(keySize);
  return keyPairGenerator.genKeyPair();
 }
 public static byte[] encryptAES(PublicKey publicKey , byte [] AESkeybytes ) throws Exception {
  Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
  cipher.init(Cipher.ENCRYPT_MODE, publicKey);
  //System.out.println("Cipher inited ");
  byte[] encryptedAESkeybytes = cipher.doFinal(AESkeybytes);
  // System.out.println("encryptedTextByte is "+ encryptedAESkeybytes+ "with length of " +encryptedAESkeybytes.length+
"bytes");
  return encryptedAESkeybytes;
   }

 public static  SecretKey decryptAES(PrivateKey privateKey , byte [] encryptedAESkeybytes) throws Exception {
  Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
  cipher.init(Cipher.DECRYPT_MODE,privateKey );
  // System.out.println("DEcryptText is "+ encryptedAESkeybytes);
  byte[] keybytesALL = cipher.doFinal(encryptedAESkeybytes);
  byte[] keybytes= Arrays.copyOfRange(keybytesALL, 0, 32);
  SecretKey key = new SecretKeySpec ( keybytes, "AES" );
  //System.out.println("AES Key is "+ key+"with length of " +key.getEncoded().length+ "bytes" );
  return key;
 }}
```

AES Class:

```
import java.security.NoSuchAlgorithmException;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

public class AES {

        public static SecretKey buildKey() throws NoSuchAlgorithmException {
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(256);
        SecretKey secretKey = keyGen.generateKey();
        return secretKey;
        }

        public static byte[] encryptText(SecretKey secretKey , String message ) throws Exception {
          Cipher cipher = Cipher.getInstance("AES");
          //System.out.println("encryptText is "+ message);
          cipher.init(Cipher.ENCRYPT_MODE, secretKey);
          // System.out.println("Cipher inited ");
          byte[] textbyte = cipher.doFinal(message.getBytes());
          //System.out.println("encryptedTextByte is "+ textbyte);
          //System.out.println("encryptedTextByte SIZE is "+ textbyte.length);
          // String text= textbyte.toString();

          return textbyte;

          //return cipher.doFinal(message.getBytes());
        }

        public static String decryptText(SecretKey secretKey , byte [] ciphertext) throws Exception {
          Cipher cipher = Cipher.getInstance("AES");
          cipher.init(Cipher.DECRYPT_MODE,secretKey );
          // System.out.println("DEcryptText is "+ ciphertext);
          byte[] textbyte = cipher.doFinal(ciphertext);
          // System.out.println("DEcryptedTextByte is "+ textbyte);
          // System.out.println("DEcryptedTextByte SIZE is "+ textbyte.length);
          //String text= textbyte.toString();
          //System.out.println("DEcrypted Text is "+ text);
          return new String(textbyte) ;
        }

}
```

A.2 Πηγαίος κώδικας για το 8.4

Client Class:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
```

```
import java.net.Socket;
import java.nio.file.Files;
import java.security.KeyPair;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import java.util.Calendar;

public class ClientChat extends JFrame implements ActionListener {
  static Socket con;
  JPanel panel;
  JTextField NewMsg;
  JTextArea ChatHistory;
  JButton Send;

  String line;
  BufferedReader br;
  String response;
  BufferedReader is ;
  OutputStream os;
  PublicKey ServerpublicKey;
  static PublicKey Clientpublickey ;
  static PrivateKey Clientprivatekey;
            static String IP="127.0.0.1";

  public ClientChat() throws Exception {
    panel = new JPanel();
    NewMsg = new JTextField();
    ChatHistory = new JTextArea();
    Send = new JButton("send");
    this.setSize(500, 500);
    this.setVisible(true);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    panel.setLayout(null);
    this.add(panel);
    ChatHistory.setBounds(20, 20, 450, 360);
    panel.add(ChatHistory);
    NewMsg.setBounds(20, 400, 340, 30);
    panel.add(NewMsg);
    Send.setBounds(375, 400, 95, 30);
    panel.add(Send);

    Send.addActionListener(this);
    this.setTitle("Secure Client");
    //System.out.println("Creation of McEliece keys started");
    GeTtimestamp("Creation of keys started");
    long startTime = System.nanoTime();
    //Generate local Mceliece keys public private
    KeyPair keypair = createkeypair();
    long endTime = System.nanoTime();
    long duration = (endTime - startTime)/1000000 ;
    //System.out.println("Creation of McEliece keys END");
    GeTtimestamp("Creation of McEliece keys END  "+duration+"  milliseconds");
    final int lengthofpublic =keypair.getPublic().getEncoded().length;
    //Write the keys locally
    WriteKeystoFile(keypair);

    //conn = new Socket(InetAddress.getLocalHost(), 2000);
    con = new Socket(InetAddress.getByName(IP),2000);
    ChatHistory.setText("Connected to Server");
    PublicKey Clientpublickey = keypair.getPublic();
    PrivateKey Clientprivatekey = keypair.getPrivate();
                        //ChatHistory.setText(ChatHistory.getText() + '\n' + "creation of McEliece keypair finished");
                        //System.out.println("CLIENT MAIN: Public key " + Clientpublickey) ;

                        //System.out.println("Exchange of PUBLIC keys started ");
                        GeTtimestamp("Exchange of McEliece PUBLIC keys started");
                        long startTime1 = System.nanoTime();
    //Send public key to Server
                        sendclientpubKey();
```

```
                    //ChatHistory.setText(ChatHistory.getText() + '\n' + "Sent public key to SERVER ");
 //Get servers public key
                    receiveserverpubKey(lengthofpublic);
              //Reconstruct Server Public key from file as downloaded from Server
                    //System.out.println("Exchange of PUBLIC keys FINISH ");
                    long endTime1 = System.nanoTime();
                    long duration1 = (endTime1 - startTime1)/1000000 ;

                    PublicKey Serverpublic = McEliece.PublicKeyReader("/home/antony/Client/Serverpublic.key");
                    GeTtimestamp("Exchange of McEliece PUBLIC keys FINISH  "+duration1+"  milliseconds");
                    GeTtimestamp("Exchange of McEliece/AES key START");
                    long startTime2 = System.nanoTime();
                    receiveSecret(Clientprivatekey );
                    long endTime2 = System.nanoTime();
                    long duration2 = (endTime2 - startTime2)/1000000 ;
                    GeTtimestamp("Exchange of McEliece/AES key END  "+duration2 +"  milliseconds");
                    //System.out.println("CLIENT MAIN: SERVER Public key is " + Serverpublic) ;

                    //ChatHistory.setText(ChatHistory.getText() + '\n' + "Received SERVER public key ");

 ChatHistory.setText(ChatHistory.getText() + '\n' + "McEliece-AES chat Secured on APP Level started ");
 while (true) {
  try {
            SecretKey secret = getAESkeyfromfile();
            //System.out.println("CLIENT MAIN: Key is "+ secret) ;
            DataInputStream dis = new DataInputStream(con.getInputStream());
            //System.out.println("CLIENT MAIN: DataInputStream OPENED") ;
            int length = dis.readInt();
    byte[] cyphertextbytes = new byte[length];
    dis.readFully(cyphertextbytes,0,cyphertextbytes.length);
    //System.out.println("CLIENT MAIN: Read something") ;
    //System.out.println("CLIENT MAIN: encrypted in bytes:" + cyphertextbytes) ;
    ///byte[] cyphertextbytes = cyphertext.getBytes();
    String decryptedcyphertext =AES.decryptText(secret, cyphertextbytes) ;
    //System.out.println("CLIENT MAIN: decrypted :" + decryptedcyphertext) ;

    //simple implementation String line = is.readLine();
    ChatHistory.setText(ChatHistory.getText() + '\n' + "Server:"
       +new String(decryptedcyphertext));
    // GeTtimestamp();
   } catch (Exception e1) {
    ChatHistory.setText(ChatHistory.getText() +'\n'
       + "Message Receive fail:Network Error");
    try {
      Thread.sleep(3000);
      System.exit(0);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
   }
  }
}

@Override
public void actionPerformed(ActionEvent e) {

 if ((e.getSource() == Send) && (NewMsg.getText() != "")) {
            ChatHistory.setText(ChatHistory.getText() + '\n' + "Me:"
     + NewMsg.getText());
   try
   {
            SecretKey secret = getAESkeyfromfile();
            DataOutputStream dos = new DataOutputStream(
                                                con.getOutputStream());
                                String text =NewMsg.getText();
                                byte[] textbytes =                  text.getBytes();
                                byte[] cyphertextbytes=AES.encryptText(secret, text);
                                dos.writeInt(cyphertextbytes.length);
                                dos.write(cyphertextbytes);
                                System.out.println( "CLIENT MAIN:Text to SEND is " +NewMsg.getText());
                                System.out.println( "CLIENT MAIN: Encrypted Text bytes to SEND is " +cyphertextbytes);

   } catch (Exception e1) {
    ChatHistory.append(ChatHistory.getText() + '\n'
       + "Message sending fail:Network Error");
```

```
   }


   NewMsg.setText("");
  }
}

public static KeyPair createkeypair() throws NoSuchAlgorithmException{
                        KeyPair keypair = McEliece.generateKeyPair();
                        PublicKey ClientpublicKey = keypair.getPublic();
                        PrivateKey ClientprivateKey = keypair.getPrivate();
                        //System.out.println("Client PUB KEY IS:"+ ClientpublicKey) ;
                        return keypair;
        }


public void  WriteKeystoFile(KeyPair keypair) throws IOException {
        PublicKey ClientpublicKey = keypair.getPublic();
        PrivateKey ClientprivateKey = keypair.getPrivate();
                        FileOutputStream fosPub = new FileOutputStream("/home/antony/Client/Clientpublic.key");
                        fosPub.write(ClientpublicKey.getEncoded());
                        fosPub.close();
                        FileOutputStream fosPriv = new FileOutputStream("/home/antony/Client/Clientprivate.key");
                        fosPriv.write(ClientprivateKey.getEncoded());
                        fosPriv.close();
        }

public static SecretKey getAESkeyfromfile() throws IOException {
        byte []keybyte = new byte[32];
        FileInputStream fin = new FileInputStream("/home/antony/Client/secret.key");
        fin.read(keybyte);
        SecretKey secret = new SecretKeySpec(keybyte,"AES");
        //System.out.println("CLIENT SECRET IS:" + secret );
        fin.close();
        return secret;
}


public void sendclientpubKey() throws IOException, ClassNotFoundException{


        //try {
                        Socket clientSocket = new Socket(IP,1979);

                        //System.out.println("Client Socket 2 opened");
                        File clientpubkeyfile = new File ("/home/antony/Client/Clientpublic.key");
 byte [] mybytearray = new byte [(int)clientpubkeyfile.length()];
 FileInputStream fis = new FileInputStream(clientpubkeyfile);
 BufferedInputStream bis = new BufferedInputStream(fis);
 bis.read(mybytearray, 0, mybytearray.length);
 OutputStream os = clientSocket.getOutputStream();
 os.write(mybytearray, 0, mybytearray.length);
 //System.out.println("Sending " + "/home/antony/Client/Clientpublic.key" + "(" + mybytearray.length + " bytes)");
                        os.flush();
                        if (bis != null) bis.close();
 if (os != null) os.close();
 if (clientSocket!=null) clientSocket.close();

 // catch (Exception e1) {}
                        //System.out.println("Client Socket 2 closed");
                        //System.out.println("SENT Client Public key is"+mybytearray);

}

        public static void receiveserverpubKey(int lengthofpublic ) throws IOException, ClassNotFoundException{

        Socket clientSocket = new Socket(IP,1978);
        //System.out.println("Client Socket 1 opened");
        byte [] mybytearray =new byte[lengthofpublic];
          InputStream is = clientSocket.getInputStream();
          FileOutputStream fos = new FileOutputStream("/home/antony/Client/Serverpublic.key");
          BufferedOutputStream bos = new BufferedOutputStream(fos);
          try {
          int bytesRead = is.read(mybytearray,0,mybytearray.length);

          int current = bytesRead;
          do {
```

```
                bytesRead =
                 is.read(mybytearray, current, (mybytearray.length-current));
                if(bytesRead >= 0) current += bytesRead;
               } while(bytesRead<-1);

               bos.write(mybytearray, 0 , current);
               bos.flush();
                } catch (Exception e1) {}
               if (fos != null) fos.close();
               if (bos != null) bos.close();
               if (clientSocket != null) clientSocket.close();

               //System.out.println("File " + "/home/antony/Client/ServerPublic.key"+ " downloaded " + mybytearray.length + " bytes
read)");
               File file = new File("/home/antony/Client/ServerPublic.key");
               //System.out.println("File " +"/home/antony/Client/ServerPublic.key size is"+ file.length()+ " bytes ");
                        //System.out.println("RECEIVED Server Public key is"+mybytearray);
              //System.out.println("Client Socket 1 closed");
  }

              public static void receiveSecret(PrivateKey myRSAprivate ) throws Exception{

                        Socket clientSocket = new Socket(IP,1980);
                        //System.out.println("Client Socket 3 opened");
                        byte [] mybytearray =new byte[256];
                          InputStream is = clientSocket.getInputStream();
                          FileOutputStream fos = new FileOutputStream("/home/antony/Client/ENCsecret.key");
                          BufferedOutputStream bos = new BufferedOutputStream(fos);
                          int bytesRead = is.read(mybytearray,0,mybytearray.length);
                          int current = bytesRead;
                          try {
                                    do {
                            bytesRead =
                             is.read(mybytearray, current, (mybytearray.length-current));
                            if(bytesRead >= 0) current += bytesRead;
                          } while(bytesRead<-1);

                          bos.write(mybytearray, 0 , current);
                          bos.flush();
                           } catch (Exception e1) {}


                          if (fos != null) fos.close();
                          if (bos != null) bos.close();
                          if (clientSocket != null) clientSocket.close();

                         File file = new File("/home/antony/Client/ENCsecret.key");

                        //System.out.println("Client Socket 3 closed");

                        //convert encrypted AES file to bytes
                        byte[] EncrAESbyte = Files.readAllBytes(new File("/home/antony/Client/ENCsecret.key").toPath());
                        //System.out.println("encrypted AES byte array is :" +EncrAESbyte.length+"bytes");
                        byte[] AESbyte =McEliece.decryptText(myRSAprivate, EncrAESbyte);

                        //Write AES to file
                        FileOutputStream ffos = new FileOutputStream("/home/antony/Client/secret.key");
                        ffos.write(AESbyte, 0, AESbyte.length);
                        ffos.close();
                        //Get back AES key from file
                        byte[] AESbytefromfile = Files.readAllBytes(new File("/home/antony/Client/secret.key").toPath());
                        SecretKeySpec secretKeySpec = new SecretKeySpec(AESbytefromfile, "AES");

                }
              public static void GeTtimestamp(String e) {
              Calendar calendar = Calendar.getInstance();
              java.util.Date currentTimestamp = new java.sql.Timestamp(calendar.getTime().getTime());
  System.out.println(currentTimestamp+e);

              } public static void main(String[] args) throws Exception
{
  ClientChatForm chatForm = new ClientChat();
}}
```

Server Class:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.KeyPair;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.Calendar;

import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class ServerChatForm1 extends JFrame implements ActionListener {
 static ServerSocket server;
 static Socket conn;
 JPanel panel;
 JTextField NewMsg;
 JTextArea ChatHistory;
 JButton Send;

            static PrivateKey Serverprivate;
            static PublicKey ClientpublicKey1;
            static String IP="127.0.0.1";
 //DataInputStream dis;
 //DataOutputStream dos;
  String line;
  BufferedReader is ;
  public ServerChat() throws Exception {

  panel = new JPanel();
  NewMsg = new JTextField();
  ChatHistory = new JTextArea();
  Send = new JButton("send");
  this.setSize(500, 500);
  this.setVisible(true);
  setDefaultCloseOperation(EXIT_ON_CLOSE);
  panel.setLayout(null);
  this.add(panel);
  ChatHistory.setBounds(20, 20, 450, 360);
  panel.add(ChatHistory);
  NewMsg.setBounds(20, 400, 340, 30);
  panel.add(NewMsg);
  Send.setBounds(375, 400, 95, 30);
  panel.add(Send);
  this.setTitle("Secure Server");
  Send.addActionListener(this);
  Calendar calendar = Calendar.getInstance();
  java.util.Date currentTimestamp = new java.sql.Timestamp(calendar.getTime().getTime());
 //Antony Generate local McEliece keys public private
   KeyPair keypair = createkeypair();
   final int lengthofpublic =keypair.getPublic().getEncoded().length;
 //Write the keys locally
                        WriteKeystoFile(keypair);
  //server = new ServerSocket(2000, 1, InetAddress.getLocalHost());
```

```
server = new ServerSocket(2000, 1, InetAddress.getByName(IP));
//System.out.println("SERVER MAIN: Public key " + keypair.getPublic()) ;
ChatHistory.setText("Waiting for Client");
conn = server.accept();

//Create AES 256 symmetric key and store it to file
 SecretKey secret = AES.buildKey();
 WriteSecrettoFile(secret);


//System.out.println("SERVER MAIN: Public key " + keypair.getPublic()) ;

                    //byte[] encodedPublicKey = keypair.getPublic().getEncoded();
          //byte[] encodedPrivateKey = keypair.getPrivate().getEncoded();
                    //System.out.println("SERVER MAIN: createkeypair finished");
                    //ChatHistory.setText(ChatHistory.getText() + '\n' + "creation of keypair finished");
                    //System.out.println("SERVER MAIN: Public key " + keypair.getPublic()) ;


 //AntonyGet Clients public key

                    receiveclientpubKey(lengthofpublic);


                    //ChatHistory.setText(ChatHistory.getText() + '\n' + "Received Client PUBLIC KEY");
//AntonySend public key to Client
                    sendserverpubKey();
/*
                    Security.addProvider(new FlexiPQCProvider());
  //System.out.println("SERVER MAIN: Received Client Public key " + ClientPublickey) ;
                    String teststring = "hello";
                    ChatHistory.setText(ChatHistory.getText() + '\n' + "Sent public key to Client ");
                    byte[] cipher= McEliece.encryptText(keypair.getPublic(), teststring.getBytes());
                    System.out.println( "Encrypting/:"+teststring +" with my public "+ teststring.getBytes());
                    System.out.println( "Encrypted is::" + cipher);
                    byte[] decryptedbyte = McEliece.decryptText(keypair.getPrivate(), cipher);
                    String decrypted = new String(decryptedbyte);
                    System.out.println("Decrypted should be hello and it is::" +decrypted );*/


          //Send AES to Client
                    PublicKey ClientPublickey = McEliece.PublicKeyReader("/home/antony/Server/Clientpublic.key");
                    sendSecretKey(secret,ClientPublickey);

ChatHistory.setText(ChatHistory.getText() + '\n' + "Client Found");
ChatHistory.setText(ChatHistory.getText() + '\n' + "McEliece chat Secured on APP Level started ");
while (true) {
 try {

          //Read and recreate AES from file
          SecretKey secret1 = getAESkeyfromfile();
           //System.out.println("SERVER MAIN: Secret key from file is:" + secret1);
          DataInputStream dis = new DataInputStream(conn.getInputStream());
          //System.out.println("SERVER MAIN: DataInputStream OPENED") ;
          int length = dis.readInt();
  byte[] cyphertextbytes = new byte[length];
  dis.readFully(cyphertextbytes,0,cyphertextbytes.length);
  //System.out.println("SERVER MAIN: encrypted in bytes:" + cyphertextbytes) ;
  String decryptedcyphertext =AES.decryptText(secret1, cyphertextbytes) ;
  //System.out.println("SERVER MAIN: Decrypted in bytes:" + decryptedcyphertext) ;

  ChatHistory.setText(ChatHistory.getText() + '\n' + "Server:"
    + new String(decryptedcyphertext));
  GeTtimestamp();

 } catch (Exception e1)
    {
  ChatHistory.setText(ChatHistory.getText() + '\n'
    + "Message Receiveing fail:Network Error"+e1.toString());
  System.out.println("SERVER MAIN:Message Receiveing fail:Network Error"+e1.toString()) ;
  try {
   Thread.sleep(3000);
   System.exit(0);
  } catch (InterruptedException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
```

```
  }
 }
}

@Override
public void actionPerformed(ActionEvent e) {
 if ((e.getSource() == Send) && (NewMsg.getText() != "")) {
  ChatHistory.setText(ChatHistory.getText() + '\n' + "ME:"
    + NewMsg.getText());
  try {
            //Read and recreate AES from file
            SecretKey secret = getAESkeyfromfile();
            System.out.println("SERVER MAIN: key is :" +secret);

            DataOutputStream dos = new DataOutputStream(
                                                        conn.getOutputStream());
                                    String text =NewMsg.getText();
                                    //byte[] textbytes =text.getBytes();
                                    byte[] cyphertextbytes=AES.encryptText(secret, text);
                                    dos.writeInt(cyphertextbytes.length);
                                    dos.write(cyphertextbytes);
                                    //dos.flush();
                                    System.out.println("SERVER MAIN: cyphertextbytes was written to socket" ) ;

  } catch (Exception e1) {
   try {
    Thread.sleep(3000);
    System.exit(0);
   } catch (InterruptedException e2) {
    // TODO Auto-generated catch block
    e2.printStackTrace();
   }
  }

  NewMsg.setText("");
 }
}


public static KeyPair createkeypair() throws NoSuchAlgorithmException{
                        KeyPair keypair = McEliece.generateKeyPair();
                        PublicKey ServerpublicKey = keypair.getPublic();
                        PrivateKey ServertprivateKey = keypair.getPrivate();
                        //System.out.println("SERVER PUB KEY IS:"+ ServerpublicKey) ;
                        return keypair;
            }


public void  WriteKeystoFile(KeyPair keypair) throws IOException {
            byte[] ServerpublicKey = keypair.getPublic().getEncoded();
            byte[] ServerprivateKey = keypair.getPrivate().getEncoded();
            //System.out.println("SERVER PUB KEY LENGTH IS:"+ ServerpublicKey.length) ;
            //System.out.println("SERVER PRIV KEY LENGTH IS:"+ ServerprivateKey.length) ;
                        FileOutputStream fosPub = new FileOutputStream("/home/antony/Server/Serverpublic.key");
                        fosPub.write(ServerpublicKey);
                        fosPub.close();
                        FileOutputStream fosPriv = new FileOutputStream("/home/antony/Server/Serverprivate.key");
                        fosPriv.write(ServerprivateKey);
                        fosPriv.close();
                        File file1 =new File("/home/antony/Server/Serverpublic.key");
                        File file2 =new File("/home/antony/Server/Serverprivate.key");
                        double bytes=file1.length();
                        double bytes1=file2.length();
                        //System.out.println("SERVER PUB KEY LENGTH in FILE:"+ bytes);
                        //System.out.println("SERVER PRIVATE KEY LENGTH in FILE:"+ bytes1);

            }

public void  WriteSecrettoFile(SecretKey secret ) throws IOException {

                        FileOutputStream fosPub = new FileOutputStream("/home/antony/Server/secret.key");
                        fosPub.write(secret.getEncoded());
                        fosPub.close();

            }

public static SecretKey getAESkeyfromfile() throws IOException {
```

```
            byte []keybyte = new byte[32];
            FileInputStream fin = new FileInputStream("/home/antony/Server/secret.key");
            fin.read(keybyte);
            SecretKey secret = new SecretKeySpec(keybyte,"AES");
            fin.close();
            return secret;
}


public void sendserverpubKey() throws IOException, ClassNotFoundException{
            ServerSocket server1 = new ServerSocket(1978, 1, InetAddress.getByName(IP));
            //ServerSocket server = new ServerSocket(1978);

            Socket s=server1.accept();
                        //System.out.println("Server Socket 1 opened");
                        File serverpubkeyfile = new File ("/home/antony/Server/Serverpublic.key");
                        byte [] mybytearray = new byte [(int)serverpubkeyfile.length()];
 FileInputStream fis = new FileInputStream(serverpubkeyfile);
 BufferedInputStream bis = new BufferedInputStream(fis);
 bis.read(mybytearray, 0, mybytearray.length);
 OutputStream os = s.getOutputStream();
 os.write(mybytearray, 0, mybytearray.length);
 //System.out.println("Sending " + "/home/antony/Server/Serverpublic.key" + "(" + mybytearray.length + " bytes)");
 os.flush();
 if (bis != null) bis.close();
 if (os != null) os.close();
 if (s!=null) s.close();
                        server1.close();
                        //System.out.println("Server Public key is"+mybytearray);
                        //System.out.println("Server Socket 1 closed");

}
            //Try to receive  public Client key
public static void receiveclientpubKey(int lengthofpublic ) throws IOException, ClassNotFoundException{

            ServerSocket server2 = new ServerSocket(1979, 1, InetAddress.getByName(IP));
            //ServerSocket server = new ServerSocket(1979);
                        Socket s=server2.accept();
                        //System.out.println("Server Socket 2 opened");

                         byte [] mybytearray =new byte[lengthofpublic];
              InputStream is = s.getInputStream();
              FileOutputStream fos = new FileOutputStream("/home/antony/Server/Clientpublic.key");
              BufferedOutputStream bos = new BufferedOutputStream(fos);
              int count=0;
                        while (( count = is.read(mybytearray)) > 0) {fos.write(mybytearray, 0, count);}
              //int bytesRead = is.read(mybytearray,0,mybytearray.length);
              /*int current = bytesRead;
              do {
                bytesRead =
                 is.read(mybytearray, current, (mybytearray.length-current));
                if(bytesRead >= 0) current += bytesRead;
              } while(bytesRead<-1);

              bos.write(mybytearray, 0 , current);
              bos.flush();*/

              if (fos != null) fos.close();
              if (bos != null) bos.close();
              if (s != null) s.close();
               server2.close();
             // File file = new File("/home/antony/Server/Clientpublic.key");

              //System.out.println("File " + "/home/antony/Server/Clientpublic.key"
               // + " downloaded " + mybytearray.length + " bytes read");
              //System.out.println("File " + "/home/antony/Server/Clientpublic.key size is"+ file.length()+ " bytes ");
                        //System.out.println("Server Socket 2 closed");
                        //System.out.println("RECEIVED Client Public key is"+mybytearray);
}

public void sendSecretKey(SecretKey secret,PublicKey clientpublic) throws Exception{
            ServerSocket server1 = new ServerSocket(1980, 1, InetAddress.getByName(IP));
            //ServerSocket server = new ServerSocket(1978);

            Socket s=server1.accept();
                        //System.out.println("Server Socket 3 opened");
                        byte[] secretbyte = secret.getEncoded();
```

```
                        byte[] encryptedsecret = McEliece.encryptText(clientpublic, secretbyte);
                        File encryptedsecretfile = new File ("/home/antony/Server/ENCsecret.key");
                        FileOutputStream fosPub = new FileOutputStream(encryptedsecretfile);
                        fosPub.write(encryptedsecret);
                        fosPub.close();
                        byte [] mybytearray = new byte [(int)encryptedsecretfile.length()];
   FileInputStream fis = new FileInputStream(encryptedsecretfile);
   BufferedInputStream bis = new BufferedInputStream(fis);
   bis.read(mybytearray, 0, mybytearray.length);
   OutputStream os = s.getOutputStream();
   os.write(mybytearray, 0, mybytearray.length);
   //System.out.println("Sending " + "/home/antony/Server/ENCsecret.key" + "(" + mybytearray.length + " bytes)");
   os.flush();
   if (bis != null) bis.close();
   if (os != null) os.close();
   if (s!=null) s.close();
                        server1.close();
                        //System.out.println("Server Secret key is"+mybytearray);
                        //System.out.println("Server Socket 3 closed");

  }
  public static void GeTtimestamp() {
            Calendar calendar = Calendar.getInstance();
            java.util.Date currentTimestamp = new java.sql.Timestamp(calendar.getTime().getTime());
   System.out.println(currentTimestamp);
  }

  public static void main(String[] args) throws Exception {
  new ServerChatForm1();

  }
}
```

McEliece Class:

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Security;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import de.flexiprovider.api.keys.KeySpec;
import de.flexiprovider.pki.X509EncodedKeySpec;
import de.flexiprovider.pqc.FlexiPQCProvider;
import java.security.KeyFactory;
import java.security.KeyPairGenerator;


//Download all from here:  https://www.flexiprovider.de/download.html
public class McEliece{


            //Return McEliece Keypair
            public static java.security.KeyPair generateKeyPair(){

  int KeySize = 1024;
  // Security.addProvider(new FlexiCoreProvider());
  Security.addProvider(new FlexiPQCProvider());

  try
  {  KeyPairGenerator kpg = KeyPairGenerator.getInstance("McEliece","FlexiPQC");
            kpg.initialize(KeySize);
```

```
   java.security.KeyPair keypair = kpg.generateKeyPair();
   //byte[] encodedPublicKey = keypair.getPublic().getEncoded();
                //byte[] encodedPrivateKey = keypair.getPrivate().getEncoded();
                        return keypair;
 } catch (Exception e) {
   System.out.println(e.toString());
 }
            return null;

 }

            public static java.security.PublicKey PublicKeyReader(String filename) throws IOException, InvalidKeyException,
NoSuchAlgorithmException, NoSuchProviderException, java.security.spec.InvalidKeySpecException
            {

                        byte[] keyBytes = Files.readAllBytes(new File(filename).toPath());
                        KeySpec publicKeySpec = new X509EncodedKeySpec(keyBytes);
                        KeyFactory keyFactory = java.security.KeyFactory.getInstance("McEliece", "FlexiPQC");
                        PublicKey publicKey = keyFactory.generatePublic(publicKeySpec);

                        //System.out.println("******************"+publicKey);

                        return publicKey;

 }

            public static java.security.PrivateKey PrivateKeyReader(String filename) throws IOException, InvalidKeyException,
NoSuchAlgorithmException, NoSuchProviderException, java.security.spec.InvalidKeySpecException
            {

                       byte[] keyBytes = Files.readAllBytes(new File(filename).toPath());
                        KeySpec publicKeySpec = new X509EncodedKeySpec(keyBytes);
                        KeyFactory keyFactory = java.security.KeyFactory.getInstance("McEliece", "FlexiPQC");
                        PrivateKey privatekey = keyFactory.generatePrivate(publicKeySpec);

                        //System.out.println("******************"+privatekey);

                        return privatekey;

 }


            public static byte[] encryptText(java.security.PublicKey publicKey, byte[] message ) throws IOException,
InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException
            {

                        System.out.println("Encrypting text");
                        Cipher cipher = Cipher.getInstance("McEliecePKCS"); // Obtain a McEliecePKC Cipher Object
                        cipher.init(Cipher.ENCRYPT_MODE, publicKey); // Initialize the cipher
                        byte[] ciphertextBytes = cipher.doFinal(message); // Finally encrypt the message
                        // for(byte c : ciphertextBytes)
                        //System.out.println("CipherText is "+ c + "\n");
                        return ciphertextBytes;

 }
             public static  byte[] decryptText(java.security.PrivateKey privateKey , byte [] message) throws Exception {

               Cipher cipher = Cipher.getInstance("McEliecePKCS");
               cipher.init(Cipher.DECRYPT_MODE, privateKey);

               System.out.println("DEcryptText is "+ message);
               byte[] textbyte = cipher.doFinal(message);
               System.out.println("DEcryptedTextByte is "+ textbyte);
               System.out.println("DEcryptedTextByte SIZE is "+ textbyte.length);
               //String text= textbyte.toString();
               //System.out.println("DEcrypted Text is "+ text);
               return textbyte;
             }
}
```

AES Class : Η ιδια με το A.1

Παράρτημα Β

Μετρήσεις

Ακολουθούν οι μετρήσεις με τα ακόλουθα χαρακτηριστικά σε κάθε πίνακα όπου δεν υπάρχει άλλη επεξήγηση:

i. Όνομα αλγόριθμου /1: Χρόνος δημιουργίας ζεύγους κλειδιού με τον αλγόριθμο που αναφέρεται

ii. Όνομα αλγόριθμου /2: Χρόνος ανταλλαγής δημοσίου κλειδιού με τον αλγόριθμο που αναφέρεται

iii. Όνομα αλγόριθμου /3: Χρόνος ανταλλαγής συμμετρικού κλειδιού AES με τον αλγόριθμο που αναφέρεται

Οι παρακάτω μετρήσεις είναι όλες σε milli sec

Β.1 Μετρήσεις στο σύστημα Windows 7

RSA/ 1	RSA/ 2	RSA/ 3
594	1018	315
467	1016	310
318	1000	306
361	1013	309
317	1015	309
289	1014	316
416	1015	291
319	1013	308
408	1015	314
273	1021	319

McEliece/1	McEliece/2	McEliece/ 3
1029	1027	437
958	1005	435
989	1006	944
1511	1007	453
1724	1005	935
1365	1005	432
1188	1006	942
1103	1005	432
1103	1009	433
1104	1007	940
1199	1008	450

B.2 Μετρήσεις στο σύστημα Windows 10

RSA /1	RSA /2	RSA /3
492	600	307
276	549	249
320	538	237
428	551	232
372	538	234
676	538	247
320	500	245
360	548	242
392	534	233
369	540	238

McEliece / 1	McEliece /2	McEliece /3
694	525	336
944	526	340
651	526	336
809	535	350
848	515	350
633	512	336
733	524	353
633	544	335
608	548	348
735	535	335
901	513	368

B.3 Μετρήσεις -μέσοι όροι

	Key generation time	Key exchange time	AES key exchange time
RSA Win7	376,2	1014	309,7
RSA Win10	400,5	543,6	246,4
McEliece Win7 1024	750,00	1008,18	409,00
McEliece Win10	1206,64	527,55	344,27
McEliece Win7 2048	3542	1033	409
McEliece Win7 4096	14890	1035	458
McEliece Win7 8192	84474	1085	1112