Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Μελέτη Πρωτοκόλλων Ανταλλαγής και Εγκαθίδρυσης Κρυπτογραφικών Κλειδιών**
Master Thesis Title	**Study of key exchange and key establisment protocols**
Ονοματεπώνυμο Φοιτητή	**ΣΤΥΛΙΑΝΟΣ ΜΑΝΤΑΣ**
Πατρώνυμο	**ΗΛΙΑΣ**
Αριθμός Μητρώου	**ΜΠΠΛ 08008**
Επιβλέπων	**ΠΑΝΑΓΙΩΤΗΣ ΚΟΤΖΑΝΙΚΟΛΑΟΥ, ΛΕΚΤΟΡΑΣ**

Ημερομηνία Παράδοσης **ΟΚΤΩΒΡΙΟΣ 2013**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
ΠΟΛΕΜΗ ΝΙΝΕΤΑ ΕΠΙΚΟΥΡΟΣ	**ΚΟΤΖΑΝΙΚΟΛΑΟΥ ΠΑΝΑΓΙΩΤΗΣ**	**ΔΟΥΛΗΓΕΡΗΣ ΧΡΗΣΤΟΣ**
ΚΑΘΗΓΗΤΡΙΑ	**ΛΕΚΤΟΡΑΣ**	**ΚΑΘΗΓΗΤΗΣ**

[Ευχαριστίες]

Η εργασία αυτή δεν θα είχε ολοκληρωθεί ποτέ χωρίς την πολύτιμη βοήθεια ορισμένων ανθρώπων. Για το λόγο αυτό θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Παναγιώτης Κοτζανικολάου για την καθοδήγηση που μου παρείχε, για την υποστήριξη και την εμπιστοσύνη που έδειξε στο πρόσωπό μου.

Πέραν όμως από τους ανθρώπους που συνετέλεσαν άμεσα για την εργασία αυτή θα ήθελα να ευχαριστήσω τους φίλους για την συμπαράσταση και τους γονείς μου Ηλία και Μαρία για την στήριξη που μου παρείχαν, οικονομική και συναισθηματική, κατά την διάρκεια των σπουδών μου.

Περίληψη

Στις επικοινωνίες ανάμεσα σε ανοιχτά συστήματα υπάρχουν τέσσερις κύριες κατηγορίες απειλών:

- Μη εξουσιοδοτημένη απόκτηση της πληροφορίας,
- Μη εξουσιοδοτημένη τροποποίηση της πληροφορίας,
- Πλαστοπροσωπία χρήστη,
- Αποκήρυξη της επικοινωνίας.

Ο βασικός μηχανισμός για την παροχή ασφάλειας με τα χαρακτηριστικά που περιγράφηκαν πιο πάνω είναι η κρυπτογραφία. Οι συμμετρικοί αλγόριθμοι κρυπτογραφίας είναι πολύ ασφαλείς, γρήγοροι και έχουν διαδεδομένη χρήση. Ωστόσο, υπάρχουν διάφορες ελλείψεις που σχετίζονται με σχήματα συμμετρικού κλειδιού, όπως το πρόβλημα της διανομής κλειδιού, το μεγάλο πλήθος των κλειδιών που απαιτούνται και επίσης δεν παρέχουν καμία προστασία ενάντια σε επιθέσεις της επικονωνίας. Τα παραπάνω προβλήματα λύνονται με τη χρήση ασύμμετρων αλγορίθμων ή αλλιώς δημοσίου κλειδιού σε καλά σχεδιασμένα πρωτόκολλα ασφάλειας και ανταλλαγής κλειδιού.

Υπάρχουν τρεις κυρίαρχες οικογένειες αλγορίθμων δημοσίου κλειδιού οι οποίες έχουν πρακτικό αντίκρισμα. Μπορούν να ταξινομηθούν με βάση το υπολογιστικό πρόβλημα που τις χαρακτηρίζουν:

1. Συστήματα παραγοντοποίησης ακεραίου,
2. Συστήματα διακριτού λογαρίθμου,
3. Συστήματα ελλειπτικών καμπυλών.

Κάθε μία από αυτές τις τρεις οικογένειες μπορούν να παρέχουν τους βασικούς μηχανισμούς δημοσίου κλειδιού, όπως της εγκαθίδρυσης κλειδιού, της μη αποποίησης μέσω ψηφιακών υπογραφών και της κρυπτογράφησης δεδομένων.

Ο RSA ανήκει στη κατηγορία των κρυπτογραφικών συστημάτων παραγοντοποίησης ακεραίου και είναι κατάλληλος για κρυπτογράφηση/αποκρυπτογράφηση δεδομένων, για την δημιουργία ψηφιακών υπογραφών και την επαλήθευσή τους καθώς και για την ασφαλή μεταφορά κλειδιών. Ο αλγόριθμος Diffie-Hellman Key Exchange (DHKE) ανήκει στη κατηγορία των κρυπτογραφικών συστημάτων διακριτού λογαρίθμου και αποτελεί μία εντυπωσιακή εφαρμογή του προβλήματος του διακριτού λογαρίθμου. Η θεμελιώδης αυτή τεχνική συμφωνίας κλειδιού υλοποιείται σε πολλά ανοιχτά και εμπορικά κρυπτογραφικά πρωτόκολλα όπως, στο SSH (Secure Shell) και στο IPsec (Internet Protocol Security). Στη τρίτη κατηγορία ανήκει το Elliptic Curve Diffie-Hellman ECDH το οποίο βρίσκεται σε πλήρη αναλογία με τον παραδοσιακό DHKE που αναφέραμε προηγουμένως, μόνο που εδώ πραγματοποιείται ανταλλαγή κλειδιού χρησιμοποιώντας ελλειπτικές καμπύλες. Κάποια από τα πιο γνωστά πρωτόκολλα ασφάλειας πρωτοκόλλων που χρησιμοποιούνται σήμερα κατά κόρον και χρησιμοποιούν ανταλλαγή και εγκαθίδρυση κλειδιού είναι το IPSec σε συνδυασμό με το IKE και το SSL. Ο βασικός στόχος οποιουδήποτε πρωτοκόλλου εγκαθίδρυσης επικυρωμένου κλειδιού είναι να διανέμει δεδομένα κλειδιού με ασφαλή τρόπο χρησιμοποιώντας πρωτόκολλα με έμμεση πιστοποίηση κλειδιού, με ρητή επιβεβαίωση κλειδιού κ.α.

Στα πλαίσια της διατριβής αυτής και με σκοπό τη πρακτική μελέτη κάποιων από των πιο διαδεδομένων πρωτοκόλλων κρυπτογράφησης που παρουσιάσαμε και αναλύσαμε στα προηγούμενα κεφάλαια, αναπτύξαμε μια εφαρμογή σε Java SE. Χρησιμοποιήσαμε κυρίως το πλαίσιο ανάπτυξης Java Cryptography Architecture (JCA), το περιβάλλον ανάπτυξης λογισμικού netbeans, καθώς και κρυπτογραφικές βιβλιοθήκες JCE provider, και τη συλλογή Bouncy Castle [http://bouncycastle.org/].

Τα πρωτόκολλα που καταφέραμε να υλοποιήσουμε επιτυχώς και συγκρίναμε ως προς την λειτουργικότητα και την απόδοσή τους είανι αυτά των ασύμμετρων αλγορίθμων RSA και του DH σε συνδυασμό με τη χρησή του συμμετρικού αλγορίθμου κρυπτογράφησης DES.

Abstract

Network based communications between open systems are generally subject to four major categories of threats:

- Lack of Confidentiality
- Lack of Integrity
- Masquare and man-in-the-middle
- Non-repudiation

The basic security mechanism against these attacks is the use of cryptography. The symmetric cryptographic algorithms are very secure, fast and they have widespread use. Nevertheless, there are various limitations which are related with symmetric key schemes such as the key distribution problem, the need for a large number of keys and the fact that they do not provide protection against deniability attacks. The above problems are solved by using asymmetric algorithms or otherwise known as public key algorithms in well-designed security and key exchange protocols.

In general there exist three families of public key cryptography schemes for key exchange which are of practical relevance. These can be classified by the computational problem that are characterized:

1. Integer factorization systems
2. Discrete logarithm systems
3. Elliptic curve sytems

Each of these three families can provide the basic mechanisms of public key algorithms, such as key establishment, non-repudiation information through digital signatures and data encryption.

The RSA cryptosystem belongs to integer factorization systems and is suitable to encrypt / decrypt data, to create digital signatures, to verify them and to securely exchange cryptographic keys. The Diffie-Hellman Key Exchange (DHKE) algorithm belongs to the descrete logarithm based systems and is an impressive application of the discrete logarithm problem. This basic technique of key agreement is implemented in many open and commercial cryptographic protocols such as SSH and IPsec. In the third category belongs the ECDH protocol which is full proportion of DHKE, except that here is performed an exchange key using elliptic curves. IPsec in combination with IKE and SSL are the most famous protocols of security of protocols that apply key exchangement and key establishment and are currently used routinely. The main goal of every authenticated key establishment protocol is to distribute key data safely using protocols with implicit key authentication and instant key confirmation etc.

Finally, we manage to implement and test key exchange based on the RSA and the DIFFIE-HELLMAN algorithms, combined with the use of symmetric encryption based on the DES algorithm our implementation is based on Java cryptography Architecture (JCA), JCE provider, the NetBeans development environment and crypto libraries.

Πίνακας Περιεχομένων

Περίληψη ..... - 4 -
Abstract ..... - 5 -
Πίνακας Περιεχομένων ..... - 6 -
Πίνακας Εικόνων ..... - 8 -
1. Εισαγωγή ..... - 9 -
1.1 Η ανάγκη για ασφάλεια στην επικοινωνία μέσω δικτύων ..... - 9 -
1.1.1 Απειλές ασφάλειας ..... - 10 -
1.1.2 Απαιτήσεις ασφάλειας ..... - 10 -
1.1.3 Η χρήση της κρυπτογράφησης ως μηχανισμού για την εμπιστευτικότη-τα, ακεραιότητα, αυθεντικότητα της επικοινωνίας ..... - 11 -
1.1.4 Η ανάγκη για πρωτόκολλα ανταλλαγής κλειδιού ..... - 12 -
1.2 Σκοπός και στόχος της διατριβής ..... - 13 -
1.3 Δομή της διατριβής ..... - 14 -
2. Μαθηματικό υπόβαθρο ..... - 15 -
2.1 Βασικά στοιχεία θεωρίας αριθμών ..... - 15 -
2.1.1 Οι Ακέραιοι Modulo n ..... - 15 -
2.1.2 Αλγόριθμοι στο Z_n ..... - 18 -
2.1.3 Ομάδες, Κυκλικές Ομάδες και Υποομάδες ..... - 20 -
2.2 Μονόδρομες συναρτήσεις ..... - 22 -
2.2.1 Περιγραφή ..... - 22 -
2.2.2 Αυστηρά Μονόδρομες Συναρτήσεις ..... - 22 -
2.2.3 Μονόδρομες Συναρτήσεις Κρυφής Εισόδου ..... - 23 -
2.2.4 Ένα Παράδειγμα Μονόδρομης Συνάρτησης ..... - 23 -
2.3 Συναρτήσεις κατακερματισμού ..... - 23 -
2.3.1 Κίνηρο:Υπογραφή Μεγάλων Μηνυμάτων ..... - 23 -
2.3.2 Απαιτήσεις Ασφαλείας των Συναρτήσεων Κατακερματισμού ..... - 25 -
2.3.3 Μονόδρομες ..... - 26 -
2.3.3 Ασθενής αντίσταση σε συγκρούσεις ..... - 26 -
2.3.4 Ισχυρή αντίσταση σε συγκρούσεις ..... - 27 -
2.4 Το πρόβλημα του διακριτού λογαρίθμου ..... - 28 -
2.4.1 Το Πρόβλημα του Διακριτού Λογαρίθμου σε Πρωταρχικά Σώματα ..... - 29 -
2.4.2 Το Γενικευμένο Πρόβλημα Διακριτού Λογαρίθμου ..... - 29 -
2.5 Το πρόβλημα της παραγοντοποίησης ..... - 31 -
2.6 Κρυπτογραφικός αλγόριθμος (μυστικού / δημόσιου κλειδιού) ..... - 32 -
2.6.1 Μηχανισμοί Ασφάλειας ..... - 33 -
2.6.2 Το Παραμένων Πρόβλημα: Η Αυθεντικότητα των Δημοσίων Κλειδιών ..... - 33 -
2.6.3 Σημαντικοί Αλγόριθμοι Δημοσίου Κλειδιού ..... - 34 -
2.7 Διαχείριση κλειδιών - το πρόβλημα της ανταλλαγής κλειδιού - το πρόβλημα της πιστοποίησης κλειδιού ..... - 34 -
2.7.1 Διαχείριση Κλειδιών ..... - 34 -
2.7.2 Δημόσιοι κατάλογοι ..... - 35 -
2.7.3 Ψηφιακές υπογραφές ..... - 35 -
3. Κατηγορίες πρωτοκόλλων ανταλλαγής κλειδιού ..... - 37 -
3.1 Εδραίωση κλειδιού με συμμετρική κρυπτογράφηση ..... - 37 -
3.1.1 Διανομή κλειδιού με συμμετρικές τεχνικές ..... - 38 -
3.1.2 Μεταφορά κλειδιού με συμμετρική κρυπτογράφηση ..... - 39 -
3.1.3 Συμφωνία κλειδιού με συμμετρική κρυπτογράφηση ..... - 39 -
3.2 Εδραίωση κλειδιού με κρυπτογραφία δημόσιου κλειδιού ..... - 40 -
3.2.1 Μεταφορά κλειδιού με τεχνικές δημοσίου κλειδιού ..... - 40 -
3.2.2 Συμφωνία κλειδιού με τεχνικές δημόσιου κλειδιού ..... - 41 -

3.3 Ανταλλαγή κλειδιού με RSA ..... - 43 -
3.3.1 Πλεονεκτήματα του RSA ..... - 44 -
3.3.2 Πιθανές επιθέσεις στον RSA ..... - 44 -
3.4 Εγκαθίδρυση κλειδιού με DH ..... - 45 -
3.5 Παραλλαγές DH ..... - 47 -
3.5.1 Ο Diffie-Hellman Ανταλλαγής Κλειδιού με Ελλειπτικές Καμπύλες ..... - 47 -
3.5.2 Σύγκριση του DH με τον ECDH ..... - 48 -
3.5.3 Εφήμερο DH ..... - 49 -
3.5.4 Στατικό DH ..... - 49 -
4. Μηχανισμός συμφωνίας κλειδιού σε γνωστά πρωτόκολλα ασφαλείας ..... - 51 -
4.1 Οι Στόχοι της Συμφωνίας Κλειδιού ..... - 51 -
4.2 Το πρωτόκολλο IPSec ..... - 52 -
4.2.1 Επικεφαλίδα αυθεντικοποίησης του πρωτοκόλλου IP (authentication header - AH) - 53 -
4.2.2 Ενθυλάκωση ωφέλιμου φορτίου ασφάλειας (ESP) ..... - 53 -
4.2.3 Internet Key Exchange (IKE) ..... - 54 -
4.2.4 Το Πρωτόκολλο SSL ..... - 55 -
4.2.5 Η Χειραψία του Πρωτοκόλλου SSL ..... - 56 -
4.2.6 Η Ανταλλαγή Δεδομένων στο Πρωτόκολλο SSL ..... - 58 -
5. Πρακτική μελέτη πρωτοκόλλων ανταλλαγής κλειδιού ..... - 59 -
5.1 Περιβάλλον Εργασίας ..... - 59 -
5.2 Τρόπος λειτουργίας εφαρμογής client/server ..... - 59 -
5.3 Δοκιμές ..... - 63 -
6. Συμπεράσματα ..... - 72 -
Βιβλιογραφικές Αναφορές ..... - 74 -
Γλωσσάρι ..... - 76 -
Παράρτημα Α ..... - 77 -

Πίνακας Εικόνων

Εικόνα 1. Βασική αρχή κρυπτογράφησης συμμετρικού κλειδιού[31]. - 12 -
Εικόνα 2. Ανασφαλής προσέγγιση υπογραφής μεγάλων μηνυμάτων[31]. - 24 -
Εικόνα 3. Υπογράφοντας μεγάλα μηνύματα με μία συνάρτηση κατακερματισμού[31]. - 25 -
Εικόνα 4. Βασικό πρωτόκολλο ψηφιακής υπογραφής με συνάρτηση κατακερματισμού[31]. - 25 -
Εικόνα 5. Οι τρεις ιδιότητες ασφαλείας των συναρτήσεων κατακερματισμού[31]. - 26 -
Εικόνα 6. Επίθεση αντικατάστασης[30]. - 27 -
Εικόνα 7. Επίθεση αντικατάστασης δύο μηνυμάτων[30]. - 28 -
Εικόνα 8. Βασικό πρωτόκολλο για κρυπτογράφηση δημοσίου κλειδιού[30]. - 32 -
Εικόνα 9. Το πρωτόκολλο Bellare και Rogaway[26]. - 38 -
Εικόνα 10. Πρωτόκολλο απλής μεταφοράς κλειδιού[26]. - 39 -
Εικόνα 11. Απλή μεταφορά κλειδιού με πρόσκληση – απάντηση[26]. - 39 -
Εικόνα 12. Συμφωνία κλειδιού με χρονοσφραγίδες[26]. - 39 -
Εικόνα 13. Συμφωνία κλειδιού με πρόσκληση – απάντηση[26]. - 40 -
Εικόνα 14. Ένα πρωτόκολλο μεταφοράς κλειδιού[28]. - 41 -
Εικόνα 15. Μεταφορά κλειδιού με αυθεντικοποίηση οντότητας[28]. - 41 -
Εικόνα 16. Συμφωνία κλειδιού με αμοιβαία αυθεντικοποίηση[29]. - 42 -
Εικόνα 17. Το πρωτόκολλο STS[28]. - 42 -
Εικόνα 18. Diffie-Hellman Key Exchange – DHKE[6]. - 46 -
Εικόνα 19. Παράδειγμα εκτέλεσης στο DHKE[6]. - 46 -
Εικόνα 20. Ο DH Ανταλλαγής Κλειδιού με Ελλειπτικές Καμπύλες – ECDH[11]. - 47 -
Εικόνα 21. Παράδειγμα εκτέλεσης πρωτοκόλλου ECDH[3]. - 48 -
Εικόνα 22. Εφήμερο DH[3]. - 49 -
Εικόνα 23. Στατικό DH[3]. - 49 -
Εικόνα 24. Η μορφή του πρωτοκόλλου της επικεφαλίδας αυθεντικοποίησης (AH) στο IPSec[22]. - 53 -
Εικόνα 25. Μορφή ενθυλάκωσης ωφέλιμου φορτίου ασφάλειας (ESP) των περιεχομένων ενός πακέτου IP[23]. - 54 -
Εικόνα 26. Η θέση του SSL στο ISO/OSI[29]. - 56 -
Εικόνα 27. Παράδειγμα εκκίνησης της εφαρμογής Client/Server. - 60 -
Εικόνα 28. Το παράθυρο του client. - 60 -
Εικόνα 29. Το παράθυρο του client με πληροφορίες κλειδιών. - 61 -
Εικόνα 30. Το παράθυρο του server. - 61 -
Εικόνα 31. Τα αποτελέσματα του αλγορίθμου κατά τη διαδικασία κρυπτογράφησης και αποστολής του μηνύματος. - 61 -
Εικόνα 32. Τα αποτελέσματα της διαδικασίας αποκρυπτογράφησης στον server. - 62 -
Εικόνα 33. Παραγωγή κλειδιών RSA σε JCA. - 63 -
Εικόνα 34. Βήματα εκτέλεσης κρυπτογραφημένης επικοινωνίας μεταξύ client/server με τον αλγόριθμο RSA. - 65 -
Εικόνα 35. Βήματα εκτέλεσης κρυπτογραφημένης επικοινωνίας μεταξύ client/server με το πρωτό- κολλο RSA-DES. - 67 -
Εικόνα 36. Βήματα εκτέλεσης κρυπτογραφημένης επικοινωνίας μεταξύ client/server με το πρωτό-κολλο DH-DES. - 69 -
Εικόνα 37. Πίνακας αποτελεσμάτων σεναρίων εκτέλεσης των πρωτοκόλλων DES, RSA-DES, DH-DES. - 70 -

1. Εισαγωγή

Στη παρούσα εργασία θα περιγράψουμε και θα αναλύσουμε μερικές από τις πιο γνωστές μεθόδους ασύμμετρης κρυπτογραφίας (ή αλλιώς κρυπτογραφίας δημοσίου κλειδιού) και των πρωτοκόλλων που υποστηρίζουν εκείνες τις διαδικασίες που οδηγούν στη πρακτική και ασφαλή εφαρμογή των μεθόδων της ασύμμετρης και της συμμετρικής κρυπτογραφίας. Τα πρωτόκολλα που υλοποιούν με ασφάλεια την κρυπτογραφημένη επικοινωνία δύο οντοτήτων και κάνουν χρήση των ασύμμετρων αλγορίθμων κρυπτογράφησης, ονομάζονται *πρωτόκολλα ανταλλαγής και εγκαθίδρυσης κρυπτογραφικών κλειδιών*. Τέτοια πρωτόκολλα είναι το *πρωτόκολλο διανομής κλειδιού (key distribution), το πρωτόκολλο μεταφοράς (key transport)* και το πρωτόκολλο *ανταλλαγής κλειδιού (key agreement)*. Τα κλειδιά που ανταλλάσσονται μπορεί να είναι κλειδιά μίας εφαρμογής που χρησιμοποιεί συμμετρική κρυπτογραφία ή μπορεί να είναι κλειδιά της ίδιας της ασύμμετρης κρυπτογραφίας.

Για να μπορέσουμε όμως να φτάσουμε στο σημείο να κατανοήσουμε τις περίπλοκες μαθηματικές δομές πάνω στις οποίες στηρίζονται και λειτουργούν τα παραπάνω πρωτόκολλα, θα παρουσιάσουμε αναλυτικά το μαθηματικό υπόβαθρο που απαιτείται για την κατανόηση των ασύμμετρων αλγορίθμων και των μαθηματικών προβλημάτων πάνω στα οποία στηρίζουν οι αλγόριθμοι αυτοί τη «δύναμη» τους.

Η διαδικασία απόκτησης ενός μυστικού κλειδιού μεταξύ δύο ή περισσότερων οντοτήτων, μέσω ενός μη ασφαλούς καναλιού επικοινωνίας, ονομάζεται εγκαθίδρυση κλειδιού (key establishment). Στη περίπτωση που χρησιμοποιείται σε κάθε επικοινωνία ένα διαφορετικό τέτοιο κλειδί, τότε το κλειδί ονομάζεται κλειδί συνόδου (session key). Είναι πολύ σημαντικά τα κλειδιά συνόδου γιατί μειώνουν τη ποσότητα του κρυπτογραφημένου κειμένου, μειώνουν τις συνέπειες από την αποκάλυψη ενός κλειδιού, αίρουν την ανάγκη αποθήκευσης για μεγάλο χρονικό διάστημα κρυπτογραφικών κλειδιών και δημιουργούν ανεξαρτησία μεταξύ των συνόδων επικοινωνίας και μεταξύ δικτυακών εφαρμογών. Έτσι στη πράξη, τα κλειδιά μακράς διαρκείας, είτε αυτά όπως είπαμε είναι συμμετρικά είτε είναι ζεύγη ιδιωτικών-δημόσιων κλειδιών, δε χρησιμοποιούνται ποτέ απευθείας για κρυπτογραφικές υπηρεσίες (π.χ. κρυπτογράφηση, MAC κλπ.). Αντί αυτού τα κλειδιά αυτά χρησιμοποιούνται ως κλειδιά για την ανταλλαγή κλειδιών (key encrypting keys), δηλαδή για να διευκολύνουν την ανταλλαγή κλειδιών συνόδου που είναι περιορισμένης χρονικής διάρκειας.

Οι στόχοι λοιπόν της ανταλλαγής και της εγκαθίδρυσης κλειδιού, που θα αναδείξουμε μέσα από την παρουσίαση και την ανάλυση διαφόρων πρωτοκόλλων είναι τρεις:

1. *Η Πιστοποίηση κλειδιού (key authentication)*
2. *Η Επιβεβαίωση κλειδιού (key confirmation)*
3. *Η Μοναδικότητα κλειδιού (key freshness)* – (Μοναδικότητα ως προς το ότι δεν έχει χρησιμοποιηθεί ξανά στο παρελθόν)

1.1 Η ανάγκη για ασφάλεια στην επικοινωνία μέσω δικτύων

Δεδομένου ότι οι υπολογιστές και η επικοινωνία μεταξύ τους χρησιμοποιούνται όλο και περισσότερο για αποθήκευση και ανταλλαγή πολύτιμων στοιχείων, πρέπει να λαμβάνονται μέτρα για να προστατεύονται τα στοιχεία αυτά από εσκεμμένη ή συμπτωματική τροποποίηση ή κακή χρήση. Τα στοιχεία που πρέπει να προστατευθούν ποικίλλουν, αναφορικά με τη φύση τους. Μπορεί να συνίστανται σε πόρους συστήματος που ανήκουν σε παροχείς υπηρεσιών, οι οποίοι προσφέρουν πρόσβαση σε βάσεις δεδομένων, πληροφοριακά συστήματα ή, γενικώς, υπολογιστικές υπηρεσίες που περιλαμβάνουν πληροφορίες που αποθηκεύονται και ανταλλάσσονται με τη βοήθεια υπολογιστών.

Η ασφάλεια είναι χρήσιμη τόσο για τους χρήστες των υπολογιστικών συστημάτων όσο και για τους παροχείς υπολογιστικών υπηρεσιών. Η κλασική μέθοδος επίτευξης της ασφάλειας εξυπηρετεί καλύτερα τα συμφέροντα των παροχών συστημάτων παρά αυτά των χρηστών. Πρώτον, οι χρήστες είναι συνήθως αυστηρά περιορισμένοι σε υπηρεσίες και πόρους συστήματος για τους οποίους έχουν πληρώσει ή για τους οποίους πιστεύεται από τους διαχειριστές των συστημάτων ότι είναι επαρκείς για τις ανάγκες των χρηστών. Δεύτερον, η πρόσβαση σε πόρους και πληροφορίες προστατεύεται με μηχανισμούς που βασίζονται σε συνθηματικά, οι οποίοι προστατεύουν τα συστήματα από μη εξουσιοδοτημένους χρήστες και

κάθε χρήστη από τους υπόλοιπους, αλλά δεν παρέχουν καμία προστασία στους χρήστες απέναντι στους διαχειριστές των συστημάτων. Τρίτον, τα συστήματα δίνουν τη δυνατότητα στους διαχειριστές να καταγράψουν λεπτομερώς τις κινήσεις των χρηστών, κάτι που δίνει τη δυνατότητα προστασίας των πόρων και επακριβούς χρέωσης για τους πόρους που χρησιμοποιήθηκαν, ενέχει όμως τον κίνδυνο της παραβίασης της ιδιωτικότητας των χρηστών. Όλα αυτά οδηγούν σε κλειστά συστήματα. Από την άλλη πλευρά, τα ανοικτά συστήματα που δεν παρέχουν στους χρήστες τη δυνατότητα να δημιουργήσουν κλειστά περιβάλλοντα, δεν μπορούν να φιλοξενήσουν τις περισσότερες εφαρμογές. Τα ανοικτά συστήματα στα οποία οι χρήστες έχουν τη δυνατότητα να απαιτούν υπηρεσίες, χωρίς χρονικούς ή άλλους περιορισμούς, πιθανώς με ανώνυμο τρόπο αν αυτό επιθυμείται, χωρίς προϋποθέσεις για εκ των προτέρων εγγραφή και χωρίς να είναι δυνατόν να καταγραφούν οι ενέργειές τους δεν είναι δυνατόν να στηριχθούν σε μηχανισμούς που βασίζονται στα συνθηματικά.

1.1.1 Απειλές ασφάλειας

Μπορούμε να ορίσουμε τέσσερις κύριες κατηγορίες απειλών για την ασφάλεια των επικοινωνιών στα ανοικτά συστήματα:

1. Μη εξουσιοδοτημένη απόκτηση της πληροφορίας μέσω παθητικής παρακολούθησης.
2. Μη εξουσιοδοτημένη τροποποίηση της πληροφορίας, π.χ. αλλαγή, αναπαραγωγή ή επαναποστολή της πληροφορίας που ανταλλάσσεται μεταξύ δύο οντοτήτων.
3. Μεταμφίεση, δηλαδή, διενέργεια πράξεων υπό ταυτότητα διαφορετική από την πραγματική. Η μεταμφίεση μπορεί να λάβει διάφορες μορφές.
4. Αποκήρυξη της επικοινωνίας (δηλαδή άρνηση συμμετοχής σε αυτή), από οποιοδήποτε από τα ενεχόμενα μέρη[30,31,32].

1.1.2 Απαιτήσεις ασφάλειας

Οι απαιτήσεις ασφάλειας που θα πρέπει να ικανοποιούνται σε ένα σύστημα είναι οι παρακάτω:

- Αυθεντικότητα (Authenticity)
- Εμπιστευτικότητα (Confidentiality)
- Ακεραιότητα (Integrity)
- Διαθεσιμότητα (Availability)

Η *αυθεντικότητα* είναι ένα από τα σημεία-κλειδιά της ασφάλειας. Η αυθεντικότητα των υποκειμένων (προσώπων, οργανισμών, τμημάτων υλικού) και των αντικειμένων (αρχείων, πληροφορίας, προγραμμάτων, κλειδιών) στα συστήματα διαχείρισης πληροφοριών είναι η βάση στην οποία στηρίζεται η υπευθυνότητα, που με τη σειρά της είναι η βάση για τη συλλογική εργασία. Οι απαιτήσεις που σχετίζονται με την ασφάλεια, όπως η ακεραιότητα των δεδομένων, ο έλεγχος πρόσβασης, η απουσία δυνατότητας αποκήρυξης της επικοινωνίας και η παρεμπόδιση της μεταμφίεσης μπορούν να καλυφθούν μόνο αν υπάρχει αξιόπιστη διακρίβωση της ταυτότητας των εταίρων.

Η *εμπιστευτικότητα* είναι επίσης σημαντικό ζήτημα για πολλές εφαρμογές. Πληροφορίες που είναι απόρρητες, όπως προσωπικά δεδομένα, ιατρικά στοιχεία κ.τ.λ. δεν πρέπει να μεταδίδονται χωρίς κρυπτογράφηση μέσα από δημόσια δίκτυα, αν η διαρροή αυτών των πληροφοριών είναι δυνατόν να οδηγήσει σε οικονομικές ή άλλου τύπου ζημιές. Σε πολλές περιπτώσεις η επικοινωνία μεταξύ δύο συστημάτων διέρχεται από πολλαπλά επικοινωνιακά κανάλια, με κυμαινόμενες δυνατότητες για επίδοξους υποκλοπείς να αντλήσουν την πληροφορία που επιθυμούν. Με την πρόοδο των δικτυακών συστημάτων αρχείων, όπως π.χ. τα NFS και CIFS, οι χρήστες πολλές φορές αγνοούν ότι μία ενέργεια που φαίνεται να εκτελείται «τοπικά» στον υπολογιστή τους, στην πραγματικότητα μετακινεί πολλά δεδομένα μέσα από το δίκτυο, από ή προς τον υπολογιστή όπου πραγματικά αποθηκεύεται το αρχείο. Η κρυπτογράφηση από άκρου εις άκρον είναι απαραίτητη στις περισσότερες περιπτώσεις, ενώ και μηχανισμοί κρυπτογράφησης μεταξύ δικτυακών στοιχείων επικοινωνίας μπορεί να είναι καλό να χρησιμοποιηθούν για προστασία από τεχνικές, όπως η ανάλυση ροής πληροφορίας.

Η *διαθεσιμότητα* είναι η εξασφάλιση της συνεχούς και αδιάλειπτης πρόσβασης στα δεδομένα/προγράμματα/υπηρεσίες του συστήματος.

Η *ακεραιότητα* σχετίζεται με την προστασία των αγαθών του συστήματος από μη εξουσιοδοτημένη τροποποίηση. Συχνά ο όρος ταυτίζεται με την ιδιότητα της *αυθεντικότητας*. Θα πρέπει εδώ να σημειώσουμε ότι εάν ένα μήνυμα, κατά την μεταφορά του, τροποποιηθεί εσκεμμένα από κάποιον τρίτο, τότε το μήνυμα εκτός από την ακεραιότητα, χάνει επίσης την αυθεντικότητα του.

Θα πρέπει ωστόσο να σημειώσουμε ότι η εμπιστευτικότητα προϋποθέτει την αυθεντικότητα. Ο κάθε ένας πρέπει να είναι σίγουρος για την ταυτότητα αυτού με τον οποίο ανταλλάσσει ή μοιράζεται κλειδιά για την επίτευξη της εμπιστευτικότητας. Η αυθεντικότητα είναι προαπαιτούμενο για άλλες υπηρεσίες ασφάλειας και ως εκ τούτου το κεντρικό ζήτημα στην ασφάλεια.

Συνοψίζοντας μπορούμε να εντοπίσουμε την αναγκαιότητα για:

- Δυνατότητα ισχυρής διακρίβωσης της ταυτότητας των χρηστών ή άλλων ενεργών οντοτήτων (π.χ. εκτελούμενων προγραμμάτων) με αποκεντρωμένο τρόπο και υπό τον έλεγχο του χρήστη. Δυνατότητα παροχής και επαλήθευσης αποδείξεων αυθεντικότητας και ακεραιότητας της πληροφορίας.
- Δυνατότητα εγγύησης της εμπιστευτικότητας και του απορρήτου σε ένα πολυχρηστικό περιβάλλον.

1.1.3 Η χρήση της κρυπτογράφησης ως μηχανισμού για την εμπιστευτικότητα, ακεραιότητα, αυθεντικότητα της επικοινωνίας

Ο βασικός μηχανισμός για την παροχή ασφάλειας με τα χαρακτηριστικά που περιγράφηκαν πιο πάνω είναι η κρυπτογραφία. Οι αλγόριθμοι κρυπτογραφίας χρησιμοποιούνται για δύο σκοπούς. Ο πρώτος είναι για να αποδείξει κάποιος στους υπόλοιπους ότι είναι κάτοχος κάποιου συγκεκριμένου κλειδιού. Αν ο εταίρος ή ο επαληθευτής είναι σε θέση να αποκρυπτογραφήσει δεδομένα που έχουν κρυπτογραφηθεί με χρήση του συγκεκριμένου κλειδιού, τότε θεωρείται ότι η κατοχή του κλειδιού έχει αποδειχθεί. Αν επιπρόσθετα μπορεί να εξασφαλισθεί, με άλλα μέσα, ότι κανείς άλλος χρήστης ή οντότητα δεν μπορεί να έχει στη διάθεσή του το ίδιο κλειδί, η χρήση αυτού του κλειδιού συνιστά ταυτόχρονα και σύνδεσμο προς τον χρήστη του κλειδιού. Με τον τρόπο αυτό επιτυγχάνεται η αυθεντικοποίηση. Ο δεύτερος σκοπός για τον οποίο χρησιμοποιείται η κρυπτογραφία είναι η απόκρυψη της πληροφορίας, δηλαδή η προσπάθεια να αποφευχθεί η αποκάλυψή της σε μη εξουσιοδοτημένες οντότητες.

Στην κρυπτογραφία γενικά χρησιμοποιούνται οι εξής όρο[29,30,31,32]ι:

1. *Απλό* ή *μη κρυπτογραφημένο κείμενο* (plaintext). Τα δεδομένα όπως χρησιμοποιούνται από τους ανθρώπους ή τις εφαρμογές.
2. *Κρυπτογραφημένο κείμενο* (cipher text). Τα δεδομένα σε ακατάληπτη για τους ανθρώπους ή τις εφαρμογές μορφή.
3. *Κρυπτογράφηση.* Ο μετασχηματισμός του απλού κειμένου σε κρυπτογραφημένο κείμενο.
4. *Αποκρυπτογράφηση.* Ο μετασχηματισμός του κρυπτογραφημένου κειμένου σε απλό.
5. *Κλειδί.* Μια ποσότητα πληροφορίας (σύνολο bytes) που καθορίζει τους μετασχηματισμούς που θα πραγματοποιηθούν κατά τη διαδικασία της κρυπτογράφησης ή αποκρυπτογράφησης.
6. *Χώρος μη κρυπτογραφημένων μηνυμάτων M.* Όλα τα δυνατά μηνύματα απλού κειμένου.
7. *Χώρος κρυπτογραφημένων μηνυμάτων C.* Όλα τα δυνατά μηνύματα κρυπτογραφημένου κειμένου.
8. *Χώρος κλειδιών K.* Όλα τα δυνατά κλειδιά.
9. *Οικογένεια μετασχηματισμών κρυπτογράφησης.* Μια ομάδα συναρτήσεων E_k με πεδίο ορισμού το *M* και πεδίο τιμών το *C*. Υπάρχει μία συνάρτηση για κάθε κλειδί. Η κρυπτογράφηση ενός απλού κειμένου μπορεί να γραφεί ως $E_k(m) = c$, όπου $m \in M, c \in C$.
10. *Οικογένεια μετασχηματισμών αποκρυπτογράφησης.* Μια ομάδα συναρτήσεων D_k με πεδίο ορισμού το *C* και πεδίο τιμών το *M*. Υπάρχει μία συνάρτηση για κάθε κλειδί. Η αποκρυπτογράφηση ενός κρυπτογραφημένου κειμένου μπορεί να γραφεί ως $D_k(c) = m$, όπου $m \in M, c \in C$.

Έχοντας στη διάθεσή μας τους ανωτέρω ορισμούς μπορούμε να διατυπώσουμε τους στόχους της κρυπτογραφίας με πιο τεχνικό τρόπο ως ακολούθως:

1. Εχεμύθεια
    i. Πρέπει να είναι ανέφικτο να υπολογιστεί το D_k από το c, ακόμη και αν είναι γνωστό το m.
    ii. Πρέπει να είναι ανέφικτος ο υπολογισμός του m από ένα c.
2. Αυθεντικότητα
    i. Πρέπει να είναι υπολογιστικά ανέφικτο να προσδιοριστεί το E_k από το c, ακόμη και αν είναι γνωστό το m.
    ii. Πρέπει να είναι υπολογιστικά ανέφικτο να βρεθεί ένα c', τέτοιο ώστε το $D_k(c')$ να είναι παραδεκτό μη κρυπτογραφημένο μήνυμα του συνόλου M.

Για ένα σύστημα κρυπτογραφίας είναι επίσης επιθυμητές οι κάτωθι ιδιότητες:

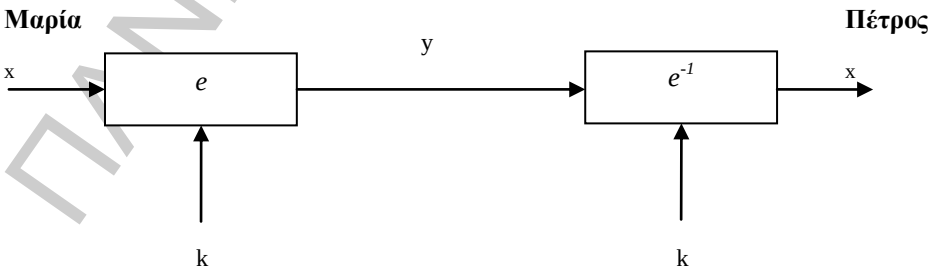
- Πρέπει να υπάρχουν αποδοτικοί αλγόριθμοι για τις λειτουργίες της κωδικοποίησης και της αποκωδικοποίησης.
- Το σύστημα πρέπει να είναι εύχρηστο.
- Η προστασία που παρέχει το σύστημα πρέπει να προϋποθέτει μόνο τη μυστικότητα των κλειδιών, όχι του αλγόριθμου.

Αναφορικά με τις κρυπτογραφικές μεθόδους μπορούμε να διακρίνουμε δύο μείζονες κατηγορίες: η πρώτη είναι οι συμμετρικοί αλγόριθμοι, όπου η κρυπτογράφηση και η αποκρυπτογράφηση γίνεται χρησιμοποιώντας το ίδιο κλειδί αλλά αντίστροφες λειτουργίες. Ο πιο διαδεδομένος αλγόριθμος συμμετρικής κρυπτογραφίας είναι ο AES που επινοήθηκε από τους Joan Daemen και Vincent Rijmen. Η δεύτερη κατηγορία είναι οι ασύμμετροι αλγόριθμοι. Ο πρώτος επινοήθηκε από τους Diffie και Hellmann το 1976. Οι αλγόριθμοι αυτοί χρησιμοποιούν διαφορετικά κλειδιά για τις λειτουργίες της κρυπτογράφησης και της αποκρυπτογράφησης. Τα κλειδιά είναι μαθηματικώς συναρτημένα μεταξύ τους, αλλά υπάρχει η πρόσθετη απαίτηση ότι «το πολύ ένα εξ αυτών να είναι δυνατόν να υπολογιστεί από το άλλο με υπολογιστικά εφικτό τρόπο, ενώ η άλλη κατεύθυνση υπολογισμού θα πρέπει να είναι υπολογιστικά ανέφικτη» [12]. Η ιδιότητα αυτή επιτρέπει να δημοσιοποιηθεί το ένα κλειδί (αυτό που υπολογίζεται βάσει του άλλου), ενώ το άλλο κλειδί τηρείται μυστικό και συνδέεται άρρηκτα με την οντότητα που προσδιορίζει. Λόγω του σχήματος λειτουργίας αυτού, οι αλγόριθμοι αυτοί πολλές φορές ονομάζονται αλγόριθμοι δημοσίου κλειδιού[29,30,31,32].

1.1.4 Η ανάγκη για πρωτόκολλα ανταλλαγής κλειδιού

Οι ασύμμετροι αλγόριθμοι ή αλλιώς δημοσίου κλειδιού, είναι πολύ διαφορετικοί από τους συμμετρικούς αλγορίθμους όπως του AES και του DES. Οι περισσότεροι αλγόριθμοι δημοσίου κλειδιού βασίζονται σε αριθμό-θεωρητικές συναρτήσεις. Αυτό είναι αρκετά διαφορετικό από τους συμμετρικούς κρυπταλγόριθμους, όπου ο στόχος συνήθως δεν είναι μία συμπαγής μαθηματική περιγραφή ανάμεσα στην είσοδο και την έξοδο.

Προκειμένου να καταλάβουμε τη βασική αρχή της ασύμμετρης κρυπτογραφίας και πως προκύπτει η ανάγκη χρήσης πρωτοκόλλων ανταλλαγής κλειδιού, ας εξετάσουμε το βασικό σχήμα συμμετρικής κρυπτογράφησης του παρακάτω σχήματος.



Εικόνα 1. Βασική αρχή κρυπτογράφησης συμμετρικού κλειδιού[31].

Ένα τέτοιο σύστημα είναι συμμετρικό και χαρακτηρίζεται από δύο ιδιότητες:

1. Το ίδιο μυστικό κλειδί χρησιμοποιείται για την κρυπτογράφηση και την αποκρυπτογράφηση.

2. Οι συναρτήσεις κρυπτογράφησης και αποκρυπτογράφησης είναι παρόμοιες (στη περίπτωση του DES είναι στην ουσία πανομοιότυπες).

Υπάρχει μία απλή αναλογία στη συμμετρική κρυπτογραφία. Ας υποθέσουμε ότι υπάρχει ένα χρηματοκιβώτιο με δυνατή κλειδαριά. Μόνο η Μαρία και ο Πέτρος έχουν από ένα κλειδί που ανοίγει τη κλειδαριά. Τη διαδικασία κρυπτογράφησης ενός μηνύματος μπορούμε να το δούμε σαν να τοποθετούμε ένα μήνυμα μέσα στο χρηματοκιβώτιο. Για να το διαβάσει ο Πέτρος θα πρέπει να το αποκρυπτογραφήσει, δηλαδή στο παράδειγμα μας, να χρησιμοποιήσει το κλειδί του, που είναι το ίδιο με αυτό της Μαρία, και να ανοίξει το χρηματοκιβώτιο. Μοντέρνοι συμμετρικοί αλγόριθμοι όπως οι AES και 3DES είναι πολύ ασφαλείς, γρήγοροι και έχουν διαδεδομένη χρήση. Ωστόσο, υπάρχουν διάφορες ελλείψεις που σχετίζονται με σχήματα συμμετρικού κλειδιού και μας οδηγούν στη χρήση και την ανάπτυξη των πρωτοκόλλων ανταλλαγής κλειδιού[9]:

- *Πρόβλημα διανομής κλειδιού*: Το κλειδί θα πρέπει να εγκαθιδρυθεί ανάμεσα στην Μαρία και τον Πέτρο χρησιμοποιώντας ένα ασφαλές κανάλι. Θυμηθείτε ότι το κανάλι επικοινωνίας για το μήνυμα δεν είναι ασφαλές, έτσι η άμεση αποστολή του κλειδιού δια μέσου του καναλιού, δεν μπορεί να γίνει.
- *Πλήθος κλειδιών*: Ακόμη και αν λύσουμε το πρόβλημα της διανομής του κλειδιού, θα πρέπει δυνητικά να συμφωνήσουμε σε ένα πολύ μεγάλο πλήθος κλειδιών. Φανταστείτε, αν κάθε ζεύγος χρηστών χρειάζεται ένα διαφορετικό ζεύγος κλειδιών, τότε σε ένα δίκτυο με *n* χρήστες, θα υπάρχουν

$$\frac{n \cdot (n-1)}{2},$$

ζεύγη κλειδιών και κάθε χρήστης θα πρέπει να αποθηκεύει *n – 1* κλειδιά με ασφάλεια. Ακόμη και σε μικρά δίκτυα επιχειρήσεων, με 2.000 χρήστες για παράδειγμα, απαιτούνται 1.999.000 ζεύγη κλειδιών που θα πρέπει να παραχθούν και να διακινηθούν μέσω ασφαλούς καναλιού.

- *Καμία προστασία ενάντια στη δολιοφθορά από τους συναλλασσόμενους*: Και οι δυο τους έχουν τις ίδιες δυνατότητες, αφού κατέχουν το ίδιο κλειδί. Ως επακόλουθο, η συμμετρική κρυπτογραφία δεν μπορεί να χρησιμοποιηθεί για εφαρμογές που θα θέλαμε να εμποδίσουμε τη δολιοφθορά είτε από τη Μαρία ή τον Πέτρο, σε αντίθεση με αυτή από ένα ξένο. Για παράδειγμα, σε εφαρμογές ηλεκτρονικού εμπορίου είναι συχνά σημαντικό να αποδείξουμε ότι, όταν εκτελείτε μία παραγγελία από τη Μαρία και στη συνέχεια αυτή το μετανιώσει, να μη μπορεί να κατηγορήσει τον Πέτρο ότι αυτός δημιούργησε τη παραγγελία. Η πρόληψη αυτή ονομάζεται «*μη αποποίηση*» και μπορεί να επιτευχθεί μόνο με την ασύμμετρη κρυπτογραφία [9].

1.2 Σκοπός και στόχος της διατριβής

Η διατριβή αυτή έχει σκοπό τη θεωρητική και πρακτική μελέτη των *πρωτοκόλλων ανταλλαγής και εγκαθίδρυσης κρυπτογραφικών κλειδιών*:

1. Τον τρόπο λειτουργίας τους.
2. Το μαθηματικό υπόβαθρο πάνω στο οποίο αναπτύχθηκαν και στηρίζουν τη «δύναμη» τους. Μαθηματικά προβλήματα τα οποία είναι υπολογιστικά ανέφικτο να επιλυθούν.
3. Την «υποχρεωτική» χρήσης τους σε ανοιχτά συστήματα επικοινωνιών για τη δημιουργία ασφαλών περιβαλλόντων.
4. Τις κατηγορίες στις οποίες διαχωρίζονται και τα χαρακτηριστικά αυτών.
5. Τα πλεονεκτήματα/μειονεκτήματα των διάφορων κατηγοριών πρωτοκόλλων.

Τέλος με την ανάπτυξη μιας εφαρμογής client/server σε τεχνολογία Java CE έχει σαν στόχο τη συγκριτική παρουσίαση πρωτοκόλλων κρυπτογράφησης και την εξαγωγή χρήσιμων συμπερασμάτων.

1.3 Δομή της διατριβής

Η διατριβή είναι οργανωμένη σε έξι κεφάλαια. Το πρώτο κεφάλαιο μας παρουσιάζει την ανάγκη για την ασφάλεια των δεδομένων σε ανοιχτά συστήματα, τις απαιτήσεις που δημιουργούνται για το σκοπό αυτό και τις λύσεις που δίνουν σήμερα τα *πρωτόκολλα ανταλλαγής και εγκαθίδρυσης κρυπτογραφικών* κλειδιών. Το δεύτερο κεφάλαιο αναφέρεται στο μαθηματικό υπόβαθρο πάνω στο οποίο οι αλγόριθμοι, που χρησιμοποιούνται σε αυτά τα πρωτόκολλα, οφείλουν τη «δύναμη». Επίσης παρουσιάζει αναλυτικά και με παραδείγματα τις μονόδρομες συναρτήσεις (one - way functions) και τις συναρτήσεις κατακερματισμού (hash functions). Το τρίτο κεφάλαιο παρουσιάζει αναλυτικά μερικά από τα πιο γνωστά και σύγχρονα πρωτόκολλα ανταλλαγής και εγκαθίδρυσης κρυπτογραφικών κλειδιών. Για κάθε μία από τις τρεις κυρίαρχες οικογένειες αλγορίθμων δημοσίου κλειδιού (συστήματα παραγοντοποίησης ακεραίου, συστήματα διακριτού λογαρίθμου, συστήματα ελλειπτικών καμπυλών) παρουσιάζεται και ένας αλγόριθμος που αποτελεί θα λέγαμε εκπρόσωπος τους. Οι αλγόριθμοι αυτοί είναι ο RSA, ο DHKE και ο ECDH αντίστοιχα. Επίσης παρουσιάζονται και κάποιες παραλλαγές τους που καλύπτουν τις αδυναμίες των τριών παραπάνω αλγορίθμων. Στο τέταρτο κεφάλαιο γίνεται παρουσίαση μερικών από τα ποίο γνωστά πρωτόκλλα ασφαλείας όπως το IPSec,το IKE και το SSL.Στο πέμτο κεφάλαιο παρουσιάζεται η υλοποίηση μίας εφαρμογής που αναπτύξαμε σε γλώσσα Java CE την οποία και χρησιμοποιούμε για δοκιμές και συγκριτική παρουσίαση των πρωτοκόλλων που υλοποιήθηκαν. Στο έκτο και τελευταίο κεφάλαιο γίνεται μία προσπάθεια εξαγωγής συμπερασμάτων για τα πλεονεκτήμα/μειονεκτήματα των διάφορων παραλλαγών και διατύπωσης μελλοντικής έρευνας στο πεδίο των *πρωτοκόλλων ανταλλαγής και εγκαθίδρυσης κρυπτογραφικών*.

2. Μαθηματικό υπόβαθρο

Στο παρόν κεφάλαιο παρουσιάζεται το μαθηματικό υπόβαθρο πάνω στο οποίο στηρίζονται διαδεδομένοι αλγόριθμοι κρυπτογράφησης δεδομένων. Τα βασικά στοιχεία της θεωρίας αριθμών και οι μονόδρομες συναρτήσεις διαδραματίζουν καθοριστικό ρόλο στη λειτουργία αυτών των αλγορίθμων. Άμεση απόρροια των μονόδρομων συναρτήσεων (ενότητα 2.3) είναι οι συναρτήσεις κατακερματισμού (ενότητα 2.4) που χρησιμοποιούνται κυρίως για υπογραφές μηνυμάτων.

Στη συνέχεια του κεφαλαίου μελετώνται εκτενώς τα μαθηματικά προβλήματα (υπολογιστικά ανέφικτο να επιλυθούν) με βάση τα οποία κατηγοριοποιούνται και στηρίζουν τις δυνατότητες τους τα *πρωτόκολλα ανταλλαγής και εγκαθίδρυσης κρυπτογραφικών κλειδιών*. Τα προβλήματα αυτά είναι:

- Το πρόβλημα της παραγοντοποίησης (ενότητα 2.6)
- Το πρόβλημα του διακριτού λογαρίθμου (ενότητα 2.5)
- Το πρόβλημα των ελλειπτικών καμπυλών (αποτελεί γενίκευση του διακριτού λογαρίθμου)

Στο τέλος του κεφαλαίου γίνεται μελέτη των λύσεων που έχουν εφαρμοστεί για την αποτελεσματική διαχείριση και χρήση των κρυπτογραφικών κλειδιών των πιο σημαντικών αλγορίθμων δημοσίου κλειδιού.

2.1 Βασικά στοιχεία θεωρίας αριθμών

Οι ορισμοί που αναφέρονται στη παρούσα ενότητα έχουν ληφθεί από γνωστές πηγές [12,15,18,30,31,32,33,34].

2.1.1 Οι Ακέραιοι Modulo n

Ορισμός 1. *Ένας αριθμός* $d \in \mathbb{Z}$ *διαιρεί τον αριθμό* $n \in \mathbb{Z}$*, και γράφουμε d | n, αν και μόνον αν υπάρχει* $c \in \mathbb{Z}$ *τέτοιος ώστε n = dc. Αν ο d δεν διαιρεί τον n, τότε γράφουμε* $d \nmid n$*. Εάν a, b είναι ακέραιοι αριθμοί με b ≥ 1, τότε υπάρχουν μοναδικοί ακέραιοι αριθμοί q, r τέτοιοι ώστε:*

$$a = qb + r,\ 0 \le r < b \qquad (1)$$

Αλγόριθμος 1.1 Ο αλγόριθμος του Ευκλείδη

είσοδος: δύο αριθμοί $a, b \in \mathbb{N}$ με $a \ge b$

1: **while** $b > 0$ **do**
2: $r \leftarrow a \bmod b$
3: $\{a, b\} \leftarrow \{b, r\}$
4: **end while**
5: $d \leftarrow a$

έξοδος: ο μέγιστος κοινός διαιρέτης d των a, b

Η σχέση (1) είναι γνωστή σαν Θεώρημα της διαίρεσης για ακέραιους αριθμούς. Ο αριθμός q είναι πηλίκο της διαίρεσης και γράφουμε q = a div b, ενώ το r είναι το υπόλοιπο της διαίρεσης και συμβολίζεται ως r = a mod b. Ένας αριθμός $c \in \mathbb{Z}$ ονομάζεται κοινός διαιρέτης των a, b αν c | a και c | b. Το σύνολο των θετικών κοινών διαιρετών των a, b είναι πεπερασμένος αριθμός, γιατί ο c δεν μπορεί να είναι μεγαλύτερος από το min{|a|, |b|}.

Ο μεγαλύτερος αριθμός που διαιρεί τόσο τον *a* όσο και το *b* λέγεται μέγιστος κοινός διαιρέτης των *a, b* και συμβολίζεται ως gcd(*a,b*). Ο μέγιστος κοινός διαιρέτης δύο αριθμών *a, b* $\in \mathbb{N}$ με *a* ≥ *b* μπορεί να υπολογιστεί από τον *αλγόριθμο του Ευκλείδη* (Αλγ. 1.1), ο οποίος βασίζεται στην εξής ιδιότητα: gcd(*a, b*) = gcd(*b, a - b*).

Παράδειγμα 1. Εφαρμόζοντας τον αλγόριθμο του Ευκλείδη για τους αριθμούς *a* = 1925 και *b* = 693, έχουμε ότι:

$$1925 = 2 \times 693 + 539$$
$$693 = 1 \times 539 + 154$$
$$539 = 3 \times 154 + 77$$
$$154 = 2 \times 77 + 0$$

και συνεπώς ο μέγιστος κοινός διαιρέτης αυτών είναι το 77. Άμεση απόρροια του αλγορίθμου του Ευκλείδη είναι το ακόλουθο αποτέλεσμα.

Ιδιότητα 1. Έστω *a, b* $\in \mathbb{N}$ και *d* = gcd(*a, b*). Τότε, υπάρχουν ακέραιοι αριθμοί *λ, μ* τέτοιοι ώστε *aλ* + *bμ* = *d*.

Παράδειγμα 2. Από τις σχέσεις του Παραδείγματος 1. μπορούμε να γράψουμε:

$$\begin{aligned} 77 &= 539 - 3 \times 154 \\ &= 539 - 3 \times (693 - 1 \times 539) \\ &= 4 \times 539 - 3 \times 693 \\ &= 4 \times (1925 - 2 \times 693) - 3 \times 693 \\ &= 4 \times 1925 - 11 \times 693. \end{aligned}$$

που εκφράζει το 77 ως γραμμικό συνδυασμό των *a* και *b*.

Αλγόριθμος 1.2 Ο επεκταμένος αλγόριθμος του Ευκλείδη

είσοδος: δύο αριθμοί $a, b \in \mathbb{N}$ με $a \geq b$

1: $\{x_2, x_1\} \leftarrow \{1, 0\}$
2: $\{y_2, y_1\} \leftarrow \{0, 1\}$
3: **while** $b > 0$ **do**
4: $\{q, r\} \leftarrow \{a \; div \; b, a \bmod b\}$
5: $\{a, b\} \leftarrow \{b, r\}$
6: $\{\lambda, \mu\} \leftarrow \{x_2 - qx_1, y_2 - qy_1\}$
7: $\{x_2, x_1\} \leftarrow \{x_1, \lambda\}$
8: $\{y_2, y_1\} \leftarrow \{y_1, \mu\}$
9: **end while**
10: $\{d, \lambda, \mu\} \leftarrow \{a, x_2, y_2\}$

έξοδος: ο μέγιστος κοινός διαιρέτης d των a, b και $\lambda, \mu \in \mathbb{Z}: \; d = a\lambda + b\mu$

Για κάθε ζευγάρι a, b $\in \mathbb{N}$, οι αριθμοί d, λ, μ που ορίζονται στην Ιδιότητα 1 υπολογίζονται μέσω του επεκταμένου αλγορίθμου του Ευκλείδη (Αλγ. 1.2), ο οποίος ουσιαστικά εφαρμόζει τη διαδικασία του Παραδείγματος 2.

Ορισμός 2. *Ένας ακέραιος αριθμός p* ≥ 2 *ονομάζεται* πρώτος *αν οι μόνοι διαιρέτες του είναι ο* 1 *και ο p, ενώ διαφορετικά ονομάζεται* σύνθετος. *Δύο αριθμοί a, b λέγονται πρώτοι μεταξύ τους αν ισχύει* gcd(*a, b*) = 1.

Οι πρώτοι αριθμοί έχουν ιδιάζουσα σημασία όχι μόνο για την κρυπτογραφία αλλά για ολόκληρο το χώρο της Θεωρίας Αριθμών. Χαρακτηριστικό είναι το ακόλουθο αποτέλεσμα, γνωστό ως *Θεμελιώδες Θεώρημα της Αριθμητικής*.

Θεώρημα 1. *Κάθε* $n \in \mathbb{Z}, n \neq 0,$ *γράφεται με μοναδικό τρόπο σαν γινόμενο πρώτων αριθμών (χωρίς να λαμβάνεται υπόψη η διάταξη των πρώτων παραγόντων του n) ως εξής*:

$$n = \pm p_1^{e_1} p_2^{e_2} \dots p_k^{e_k} \qquad (2)$$

όπου για κάθε i = 1, . . . , *k, τα* p_i *είναι πρώτοι διαφορετικοί μεταξύ τους ανά δύο, και τα* e_i *είναι μη μηδενικοί φυσικοί αριθμοί.*

Ορισμός 3. Για κάθε ακέραιο αριθμό n ≥ 1, η συνάρτηση Euler ϕ(n) ορίζεται ως το πλήθος των θετικών ακεραίων που είναι μικρότεροι ή ίσοι από τον n και πρώτοι ως προς τον n.

Για τη συνάρτηση του Euler ισχύει ϕ(1) = 1, και έχει τις ακόλουθες ιδιότητες:

1. Αν $n = p$ όπου p πρώτος αριθμός, τότε $\phi(n) = p - 1$
2. Αν $n = p^e$ όπου p πρώτος αριθμός και $e > 0$, τότε $\phi(n) = p^e - p^{e-1}$
3. Αν $n = n_1 n_2$ όπου $\gcd(n_1, n_2) = 1$, τότε $\phi(n) = \phi(n_1)\phi(n_2)$
4. Ενώ, στη γενική περίπτωση ισχύει

$$\varphi(n) = n\prod_{i=1}^{k}\left(1-\frac{1}{p_i}\right),$$

όπου το $n \in \mathbb{Z}$ δίνεται από την (2).

Ορισμός 4. *Για κάθε αριθμό $n \in \mathbb{Z}$, $n > 1$, που η ανάλυσή του σε πρώτους παράγοντες δίνεται από την (2), η συνάρτηση Möbius μ(n) ορίζεται ως εξής:*

$$\mu(n) = \begin{cases} (-1)^k, & \alpha\nu\ e_i = 1, \forall\, i = 1, \ldots, k \\ 0, & \alpha\nu\ \exists\, i = 1, \ldots, k : e_i > 1 \end{cases}$$

ενώ ισχύει μ(1) = 1. Οι συναρτήσεις των Euler και Möbius συνδέονται μέσω της σχέσης

$$\varphi(n) = \sum_{d|n}\frac{n}{d}\mu(d) = \sum_{d|n} d\ \mu\left(\frac{n}{d}\right)$$

η οποία είναι ειδική περίπτωση του ευρέως γνωστού τύπου αντιστροφής του Möbius.

Ορισμός 5. *Έστω $n \in \mathbb{Z}$ και $[a] = \{a + \lambda n : \lambda \in \mathbb{Z}\}$. Ορίζουμε στο σύνολο των ακεραίων τη σχέση*

$$a \equiv b \pmod{n} \qquad (3)$$

αν και μόνον αν n | (a - b).

Αν ισχύει η (3), τότε λέμε ότι ο *a* είναι ισοδύναμος με τον *b* modulo *n*. Η σχέση $\equiv$ είναι σχέση ισοδυναμίας καθώς ισχύουν τα εξής:

1. $a \equiv a \pmod{n}$,
2. Αν $a \equiv b \pmod{n}$, τότε $b \equiv a \pmod{n}$,
3. Αν $a \equiv b \pmod{n}$ και $b \equiv c \pmod{n}$, τότε $a \equiv c \pmod{n}$.

Το σύνολο των ακεραίων αριθμών *x* με την ιδιότητα $x \equiv a \pmod{n}$ ονομάζεται *κλάση ισοδυναμίας* ή *κλάση υπολοίπων* του *a* mod *n* και συμβολίζεται με [*a*]. Συνεπώς, $[a] = \{a + \lambda n : \lambda \in \mathbb{Z}\}$. Ένα σύνολο από *n* αντιπροσώπους, έναν από κάθε μία από τις κλάσεις υπολοίπων [0], [1], ..., [*n* – 1] ονομάζεται πλήρες σύστημα υπολοίπων mod *n*. Χρησιμοποιούμε το $\mathbb{Z}_n$ για να συμβολίσουμε το σύνολο των υπολοίπων της διαίρεσης όλων των ακεραίων με το *n*, όπου *n* είναι θετικός ακέραιος, δηλ. $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$.

Προφανώς το $\mathbb{Z}_n$ είναι ένα πλήρες σύστημα υπολοίπων mod n, όπου ο αντιπρόσωπος κάθε κλάσης ισοδυναμίας είναι ο μικρότερος θετικός ακέραιος της κλάσης. Το σύνολο $\mathbb{Z}_n^*$ περιέχει όλα τα μη – μηδενικά στοιχεία του $\mathbb{Z}_n$. Οι πράξεις της πρόσθεσης, της αφαίρεσης και του πολλαπλασιασμού ορίζονται εντός του $\mathbb{Z}_n$ και εκτελούνται modulo n.

Παράδειγμα 3. Στο $\mathbb{Z}_{28}$ έχουμε 20 + 10 = 2, 5 – 10 = 23 και 3 × 21 = 7.

Θεώρημα 2 (Euler). Εάν gcd (a, n) = 1, τότε ισχύει

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

Για πρώτο αριθμό *p* η παραπάνω σχέση γίνεται $a^{p-1} \equiv 1 \pmod{p}$, η οποία είναι γνωστή ως *(μικρό) Θεώρημα του Fermat*.

Ορισμός 6. *Έστω* $n \in \mathbb{N}$ *και* $a, b \in \mathbb{Z}$. *Κάθε ισοδυναμία της μορφής*

$$ax \equiv b \pmod{n} \tag{4}$$

ονομάζεται γραμμική ισοδυναμία mod n.

Ο ακέραιος y είναι λύση της (4) αν $ay \equiv b \pmod{n}$. Αν ο z είναι επίσης λύση, τότε θα θεωρείται διαφορετική από την y αν $y \neq z \pmod{n}$. Ορίζουμε το πλήθος των λύσεων της (4) να είναι το πλήθος των μη ισοδύναμων λύσεων αυτής.

Ορισμός 7. *Εάν* $\gcd(a,n)=1$*, η λύση b της* $ax \equiv 1 \bmod n$ *ονομάζεται αντίστροφος του a* mod *n και συμβολίζεται με* a^{-1}.

Προφανώς, ένας αριθμός *a* είναι αντιστρέψιμος modulo *n* αν και μόνον αν ισχύει $\gcd(a,n)=1$. Από τα παραπάνω προκύπτει επίσης ότι ο αντίστροφος του *a* μπορεί να υπολογιστεί με τον επεκταμένο αλγόριθμο του Ευκλείδη.

Παράδειγμα 4. Ισχύει $3^{-1} \bmod 25 = 17$ γιατί $3 \times 17 \equiv 1 \bmod 25$. Επίσης δεν υπάρχει ο $3^{-1} \bmod 27$ γιατί $\gcd(3,27) \neq 1$.

Όσον αφορά συστήματα γραμμικών ισοδυναμιών, ένα πολύ γνωστό αποτέλεσμα που χρησιμοποιείται για την επίλυση συστημάτων είναι το *Κινέζικο Θεώρημα Υπολοίπων (Chinese Remainder Theorem - CRT)*.

Θεώρημα 3 (Κινέζικο Θεώρημα Υπολοίπων). *Έστω ο ακέραιος* $n = \prod_{i=1}^{k} n_i$ *όπου* $n_1, \ldots, n_k$ *είναι σχετικά πρώτοι μεταξύ τους ανά δύο, και* $b_1, \ldots, b_k$ *τυχαίοι ακέραιοι. Το σύστημα γραμμικών ισοδυναμιών*

$$\begin{aligned} x &\equiv b_1 \pmod{n_1} \\ &\vdots \\ x &\equiv b_k \pmod{n_k} \end{aligned}$$

Έχει μοναδική λύση mod *n η οποία δίνεται από* $x = b_1 m_1' + \cdots + b_k m_k' \bmod n$, *όπου* $m_i' = m_i \left(m_i^{-1} \bmod n_i \right)$ *και* $m_i = n / n_i$.

Ορισμός 8. *Έστω η ισοδυναμία* $x^2 \equiv a \bmod n$. *Εάν η ισοδυναμία έχει λύση, τότε ονομάζεται τετραγωνική ρίζα του a mod n, και ο a ονομάζεται τετραγωνικό υπόλοιπο modulo n. Το σύνολο των τετραγωνικών υπολοίπων modulo n συμβολίζεται με* $\mathbb{Q}_n$.

Πρόταση 1. *Έστω p περιττός πρώτος αριθμός. Τότε ισχύει* $\left|\mathbb{Q}_p\right| = (p-1)/2$, *δηλαδή τα μισά στοιχεία του* $\mathbb{Z}_p^*$ *είναι τετραγωνικά υπόλοιπα* $\bmod ulo\ p$.

Πρόταση 2. *Έστω n = p·q,* όπου *p,q περιττοί πρώτοι αριθμοί. Τότε ο* $a \in \mathbb{Z}_n^*$ *είναι τετραγωνικό υπόλοιπο* $\bmod ulo\ n$ *αν και μόνον αν* $a \in \mathbb{Q}_p \cap \mathbb{Q}_q$.

Πρόταση 3. *Έστω* $n = p_1^{e_1} \cdots p_k^{e_k}$ *όπου τα* p_i *είναι διαφορετικοί περιττοί πρώτοι αριθμοί και* $e_i > 0$. *Τότε, ο* $a \in \mathbb{Q}_n$ *έχει ακριβώς* 2^k *τετραγωνικές ρίζες* $\bmod ulo\ n$.

2.1.2 Αλγόριθμοι στο Z_n

Ας υποθέσουμε ότι ο *n* είναι ένας θετικός ακέραιος. Τα στοιχεία του $\mathbb{Z}_n$ θα αναπαριστούνται από τους ακέραιους $\{0,1,2,\ldots,n-1\}$. Παρατηρείστε ότι αν $a,b \in \mathbb{Z}_n$, τότε

$$(a+b) \bmod n = \begin{cases} a+b, & \alpha\nu\ a+b < n, \\ a+b-n, & \alpha\nu\ a+b \geq n. \end{cases}$$

Η πρόσθεση και αφαίρεση κατά modulo μπορούν να εκτελεστούν χωρίς την ανάγκη μίας μακράς διαίρεσης. Ο πολλαπλασιασμός κατά modulo των a και b μπορεί να επιτευχθεί με απλό πολλαπλασιασμό των a και b σαν να ήταν ακέραιοι, και στη συνέχεια παίρνοντας το υπόλοιπο της διαίρεσης του γινομένου που πρόκυψε με το n. Οι αντίστροφοι στο $\mathbb{Z}_n$ μπορούν να υπολογιστούν χρησιμοποιώντας τον επεκταμένο αλγόριθμο του Ευκλείδη που παρουσιάσαμε πιο πάνω.

Αλγόριθμος 1.3 Υπολογισμός πολλαπλασιαστικών αντιστρόφων στο $\mathbb{Z}_n$

είσοδος: $a \in \mathbb{Z}_n$

1: Κάνε χρήση του επεκταμένου αλγορίθμου του Ευκλείδη για να βρεις ακέραιους *x* και *y* τέτοιους ώστε $ax + ny = d$, όπου $d = \gcd(a,n)$.

2: Αν $d > 1$, τότε το $a^{-1} \bmod n$ δεν υπάρχει. Διαφορετικά, επέστρεψε (*x*).

έξοδος: $a^{-1} \bmod n$, υπό την προϋπόθεση ότι υπάρχει

Η εκθετικοποίηση κατά modulo μπορεί να εκτελεστεί αποδοτικά με τον αλγόριθμο τετραγωνίζω και πολλαπλασιάζω (square-and-multiply) τον οποίο παρουσιάζουμε παρακάτω και αποτελεί πολύ σημαντικό σημείο για πολλά κρυπτογραφικά πρωτόκολλα. Μία εκδοχή του αλγορίθμου αυτού βασίζεται στην ακόλουθη παρατήρηση [15]. Ας υποθέσουμε ότι η δυαδική αναπαράσταση του *k* είναι η $\sum_{i=0}^{t} k_i 2^i$, όπου κάθε $k_i \in \{0,1\}$. Τότε,

$$a^k = \prod_{i=0}^{t} a^{k_i 2^i} = (a^{2^0})^{k_0} (a^{2^1})^{k_1} \cdots (a^{2^t})^{k_t}.$$

Αλγόριθμος 1.4 Επαναληπτικός αλγόριθμος square-and-multiply για εκθετικοποίηση στο $\mathbb{Z}_n$

είσοδος: $a \in \mathbb{Z}_n$, και ακέραιος $0 \leq k < n$ του οποίου η δυαδική αναπαράσταση είναι

$$k = \sum_{i=0}^{t} k_i 2^i$$

1: Όρισε $b \leftarrow 1$. Αν $k = 0$ τότε επέστρεψε (*b*)

2: Όρισε $A \leftarrow a$

3: Αν $k_0 = 1$ τότε όρισε $b \leftarrow a$

4: Για *i* από 1 μέχρι *t* κάνε τα παρακάτω:

4.1: Όρισε $A \leftarrow A^2 \bmod n$

4.2: Αν $k_i = 1$ τότε όρισε $b \leftarrow A \cdot b \bmod n$

5: Επέστρεψε (*b*)

έξοδος: $a^k \bmod n$

Παράδειγμα 5. Ο Πίνακας 1 δείχνει τα βήματα που περιλαμβάνονται στον υπολογισμό του $5^{596} \bmod 1234 = 1013$. Το πλήθος των πράξεων για τις βασικές πράξεις στο $\mathbb{Z}_n$ συνοψίζονται στον Πίνακα.

i	0	1	2	3	4	5	6	7	8	9
k_i	0	0	1	0	1	0	1	0	0	1
A	5	25	625	681	1011	369	421	779	947	925
b	1	1	625	625	67	67	1059	1059	1059	1013

Πίνακας 1. Υπολογισμός του $5^{596} \bmod 1234 = 1013$ **[30].**

Πράξη κατά Modulo		Πολυπλοκότητα bit
Πρόσθεση	$(a+b) \bmod n$	$O(\lg n)$
Αφαίρεση	$(a-b) \bmod n$	$O(\lg n)$
Πολλαπλασιασμός	$(a \cdot b) \bmod n$	$O((\lg n)^2)$
Αντιστροφή	$a^{-1} \bmod n$	$O((\lg n)^2)$
Εκθετικοποίηση	$a^k \bmod n, k < n$	$O((\lg n)^3)$

Πίνακας 2. Πολυπλοκότητα bit για βασικές πράξεις στο $\mathbb{Z}_n$ **[30,31].**

2.1.3 Ομάδες, Κυκλικές Ομάδες και Υποομάδες

Η ενότητα αυτή παρέχει μία επισκόπηση των βασικών αλγεβρικών δομών και των ιδιοτήτων τους.

Ορισμός 9. *Μία δυαδική πράξη * σε ένα μη – κενό σύνολο S είναι κάθε απεικόνιση από το σύνολο* $S \times S$ *στο S. Δηλαδή, η * είναι ένας κανόνας που αντιστοιχεί σε κάθε διατεταγμένο ζεύγος (s,t) με* $(s,t) \in S$ *ένα στοιχείο του S.*

Ορισμός 10. *Μία ομάδα (G,*) αποτελείται από ένα σύνολο G και μία δυαδική πράξη * στο G που ικανοποιεί τα ακόλουθα τρία αξιώματα.*

(i) *Η πράξη είναι προσεταιριστική. Δηλαδή, ισχύει* $a*(b*c) = (a*b)*c$ *για όλα τα* $a,b,c \in G$.

(ii) *Υπάρχει στοιχείο* $e \in G$, *που ονομάζεται μοναδιαίο στοιχείο, τέτοιο ώστε* $a*e = e*a$ *για κάθε* $a \in G$.

(iii) *Για κάθε* $a \in G$, *υπάρχει αντίστροφο στοιχείο* $a^{-1} \in G$ *τέτοιο ώστε* $a*a^{-1} = a^{-1}*a = e$.

Η ομάδα (G,) είναι αβελιανή ή αντιμεταθετική αν ισχύει επιπλέον ότι*

(iv) $a*b = b*a$ *για κάθε* $a,b \in G$.

Παρατηρήστε ότι για τη πράξη της ομάδας χρησιμοποιείται το σύμβολο του πολλαπλασιασμού. Εάν η ομάδα G είναι αντιμεταθετική, τότε γράφουμε *a* + *b* και *–a* αντί του $a*b$ και a^{-1} αντίστοιχα, δηλαδή χρησιμοποιούμε συμβολισμούς της πρόσθεσης.

Ορισμός 11. *Μία ομάδα G είναι πεπερασμένη εάν η |G| είναι πεπερασμένη. Το πλήθος των στοιχείων σε μία πεπερασμένη ομάδα ονομάζεται τάξη της.*

Παράδειγμα 6. Το σύνολο των ακεραίων $\mathbb{Z}$ με τη πράξη της πρόσθεσης σχηματίζουν μία ομάδα. Το ταυτοτικό στοιχείο είναι το 0 (μηδέν) και ο αντίστροφος του *a* είναι ο *–a.*

Παράδειγμα 7. Το σύνολο , με τη πράξη της πρόσθεσης modulo *n*, σχηματίζει μία ομάδα τάξης *n*. Το σύνολο με τη πράξη πολλαπλασιασμού modulo *n* δεν είναι ομάδα, αφού δεν έχουν όλα τα στοιχεία πολλαπλασιαστικούς αντίστροφους. Ωστόσο, το σύνολο $\mathbb{Z}_n^*$ είναι μία ομάδα της τάξης $\phi(n)$ κάτω από τη πράξη του πολλαπλασιασμού modulo *n*, με ταυτοτικό στοιχείο το 1 (ένα).

Ορισμός 12. *Ένα μη κενό υποσύνολο H ενός συνόλου G είναι υποομάδα του G εάν το H είναι από μόνο του μία ομάδα που ακολουθεί τη πράξη του G. Αν το H είναι υποομάδα του G και* $H \neq G$, *τότε το H ονομάζεται κατάλληλη υποομάδα του G.*

Ορισμός 13. *Μία ομάδα G είναι κυκλική αν υπάρχει ένα στοιχείο $a \in G$ τέτοιο ώστε για κάθε $b \in G$ υπάρχει ένας ακέραιος i με $b = a^i$. Ένα τέτοιο στοιχείο a ονομάζεται γεννήτορας του G.*

Πρόταση 4. *Εάν το G είναι μία ομάδα και το $a \in G$, τότε το σύνολο όλων των δυνάμεων του a σχηματίζουν μία κυκλική υποομάδα του G, που καλείται υποομάδα που παράχθηκε από το a και συμβολίζεται με $\langle a \rangle$.*

Ορισμός 14. *Ας υποθέσουμε ότι το G είναι μία ομάδα και το $a \in G$. Η τάξη του a ορίζεται να είναι ο ελάχιστος θετικός ακέραιος t τέτοιος ώστε $a^t = 1$, υπό την προϋπόθεση ότι υπάρχει ένας τέτοιος ακέραιος. Αν δεν υπάρχει τέτοιος t, τότε η τάξη του a ορίζεται ότι είναι το ∞.*

Πρόταση 5. *Έστω το G είναι μία ομάδα, και το $a \in G$ είναι ένα στοιχείο μίας πεπερασμένης τάξης t. Τότε το $|\langle a \rangle|$, το μέγεθος δηλαδή της υποομάδας που παράχθηκε από το a, είναι ίσο με t.*

Πρόταση 6 (Θεώρημα Lagrange). *Αν το G είναι πεπερασμένη ομάδα και το H είναι μία υποομάδα του G, τότε το $|H|$ διαιρεί το $|G|$. Συνεπώς, αν το $a \in G$, η τάξη του a διαιρεί το $|G|$.*

Πρόταση 7. *Κάθε υποομάδα μίας κυκλικής ομάδας G είναι επίσης κυκλική. Στη πραγματικότητα, αν το G είναι μία κυκλική ομάδα τάξης n, τότε για κάθε θετικό διαιρέτη d του n, το G περιέχει ακριβώς μία υποομάδα τάξης d.*

Πρόταση 8. *Έστω G μία ομάδα.*

(i) *Αν η τάξη του $a \in G$ είναι ίση με t, τότε η τάξη του a^k είναι $\frac{t}{\gcd(t,k)}$.*

(ii) *Αν το G είναι μία κυκλική ομάδα τάξης n και $d \mid n$, τότε το G έχει ακριβώς $\phi(d)$ στοιχεία, τάξης d. Συγκεκριμένα, το G έχει $\phi(n)$ γεννήτορες.*

Παράδειγμα 8. Θεωρείστε την πολλαπλασιαστική ομάδα $\mathbb{Z}_{19}^* = \{1, 2, \ldots, 18\}$ τάξης 18.

Η ομάδα είναι κυκλική και ένας γεννήτορας της είναι ο *a = 2.* Οι υποομάδες του $\mathbb{Z}_{19}^*$, και οι γεννήτορες τους, παρατίθενται στον παρακάτω πίνακα.

Υποομάδα	Γεννήτορες	Τάξη
{1}	*1*	*1*
{1,18}	*18*	*2*
{1,7,11}	*7,11*	*3*
{1,7,8,11,12,18}	*8,12*	*6*
{1,4,5,6,7,9,11,16,17	*4,5,6,9,16,17*	*9*
{1,2,3,...,18}}	*2,3,10,13,14,15*	*18*

Πίνακας 3. Οι υποομάδες του $\mathbb{Z}_{19}^*$ [30,31,32].

Πρόταση 9. *Έστω ότι G είναι μία κυκλική ομάδα τάξης n και ότι a είναι ένας γεννήτορας του G. Τότε για κάθε ακέραιο k που διαιρεί το n υπάρχει ακριβώς μία κυκλική υποομάδα H του G τάξης k. Η υποομάδα αυτή παράγεται από το $a^{n/k}$. Η κυκλική υποομάδα H αποτελείται ακριβώς από τα στοιχεία του $a \in G$ τα οποία ικανοποιούν τη συνθήκη $a^k = 1$. Δεν υπάρχουν άλλες υποομάδες.*

Η παραπάνω πρόταση μας δίνει μία απευθείας μέθοδο κατασκευής μίας υποομάδας από μία δοθέν κυκλική ομάδα. Το μόνο πράγμα που χρειαζόμαστε είναι ένα πρωταρχικό στοιχείο και τη πληθικότητα *n* της ομάδας. Κάποιος μπορεί τώρα να υπολογίσει το $a^{n/k}$ και να αποκτήσει ένα γεννήτορα της υποομάδας με *k* στοιχεία.

Παράδειγμα 9. Υποθέτουμε τη κυκλική ομάδα $\mathbb{Z}_{11}^*$. Γνωρίζουμε ότι το *a* = 8 είναι ένα πρωταρχικό στοιχείο της ομάδας. Αν θέλουμε να έχουμε ένα γεννήτορα *β* για μία υποομάδα τάξης 2, τότε υπολογίζουμε:

$$\beta = a^{n/k} = 8^{10/2} = 8^5 = 32768 = 10 \bmod 11$$

Μπορούμε τώρα να επιβεβαιώσουμε ότι το στοιχείο 10 όντως παράγει την υποομάδα με δύο στοιχεία: $\beta^1 = 10, \beta^2 = 100 \equiv 1 \bmod 11, \beta^3 \equiv 10 \bmod 11, \kappa\lambda\pi.$

Παρατήρηση: Φυσικά υπάρχουν πιο έξυπνοι τρόποι να υπολογίσουμε το $8^5 \bmod 11$, όπως, $8^5 = 8^2 8^2 8 \equiv (-2)(-2)8 \equiv 32 \equiv 10 \bmod 11$[13] [19].

2.2 Μονόδρομες συναρτήσεις

2.2.1 Περιγραφή

Μία μονόδρομη (one-way) συνάρτηση είναι μία μαθηματική συνάρτηση η οποία είναι σημαντικά ευκολότερη να υπολογιστεί προς τη μία κατεύθυνση (προς τα εμπρός) από ότι στην αντίθετη κατεύθυνση (την αντίστροφη). Θα μπορούσε να είναι δυνατό, για παράδειγμα, να υπολογιστεί η διαδικασία προς τα εμπρός σε δευτερόλεπτα, αλλά για να υπολογιστεί η αντίστροφή της θα μπορούσε να πάρει μήνες ή και χρόνια, εάν αυτό ήταν δυνατόν. Μία συνάρτηση κρυφής εισόδου (*trapdoor one-way function*) είναι μία μονόδρομη συνάρτηση για την οποία η αντίστροφη κατεύθυνση είναι εύκολη να υπολογιστεί αν είναι γνωστή η μυστική πληροφορία (*trapdoor*), αλλά σε αντίθετη περίπτωση είναι πολύ δύσκολος ο υπολογισμός της.

Τα κρυπτοσυστήματα δημοσίου κλειδιού βασίζονται στις μονόδρομες συναρτήσεις κρυφής εισόδου. Το δημόσιο κλειδί δίνει πληροφορίες σχετικά με τη συγκεκριμένη περίπτωση της συνάρτησης, ενώ το ιδιωτικό κλειδί δίνει πληροφορίες σχετικά με τη μυστική πληροφορία. Οποιοσδήποτε γνωρίζει αυτή τη πληροφορία μπορεί εύκολα να υπολογίσει τη συνάρτηση προς τις δύο κατευθύνσεις, αλλά σε όποιον λείπει μπορεί να υπολογίσει τη συνάρτηση μόνο προς τα εμπρός, με σκοπό να κρυπτογραφήσει και να κάνει επαλήθευση ψηφιακής υπογραφής. Σε όλα σχεδόν τα συστήματα δημοσίου κλειδιού, το μήκος του κλειδιού αντιστοιχεί στο μέγεθος της εισόδου της μονόδρομης συνάρτησης. Όσο μεγαλύτερο είναι το κλειδί, τόσο μεγαλύτερη είναι η διαφορά μεταξύ των προσπαθειών που απαιτούνται για να υπολογιστεί η συνάρτηση προς τη μία και την αντίστροφη συνάρτηση (για κάποιον που του λείπει η μυστική πληροφορία).

Όλα τα λειτουργικά κρυπτοσυστήματα βασίζονται σε συναρτήσεις που πιστεύεται ότι είναι μονόδρομες, αλλά στη πραγματικότητα καμία συνάρτηση έχει αποδειχθεί ότι είναι έτσι. Αυτό σημαίνει ότι είναι θεωρητικά δυνατό να ανακαλυφθούν αλγόριθμοι που μπορούν για κάποιες συναρτήσεις να υπολογίσουν εύκολα την αντίστροφη κατεύθυνση, χωρίς τη μυστική πληροφορία. Η εξέλιξη αυτή θα οδηγήσει κάθε κρυπτοσύστημα που βασίζεται σε μονόδρομες συναρτήσεις να είναι επισφαλές και άρα ανώφελο.

2.2.2 Αυστηρά Μονόδρομες Συναρτήσεις

Μία αμφιμονοσήμαντη συνάρτηση $f : X \to Y$ ονομάζεται αυστηρά μονόδρομη συνάρτηση αν ισχύουν τα ακόλουθα:

1. Υπάρχει μία αποδοτική μέθοδος υπολογισμού της *f(x)* για όλα τα $x \in X$.
2. Δεν υπάρχει αποτελεσματική μέθοδος υπολογισμού του *x* από τη σχέση $y = f(x)$ για όλα τα $y \in f[X]$.

Οι συναρτήσεις αυτές δεν μπορούν να χρησιμοποιηθούν με εύλογο τρόπο για κρυπτογράφηση μηνυμάτων που ακολουθούνται από αποκρυπτογράφηση αυτών. Ωστόσο μπορούν να χρησιμοποιηθούν για ταυτοποίηση και αυθεντικοποίηση, π.χ. ένας κωδικός πρόσβασης κρυπτογραφείται με μία αυστηρά μονόδρομη συνάρτηση και αποθηκεύεται σε αυτή τη μορφή. Οποιαδήποτε στιγμή απαιτείται πρόσβαση ο κωδικός παρουσιάζεται κρυπτογραφημένος και έτσι τα δύο κρυπτοκείμενα συγκρίνονται. Αυτό το σύστημα εφαρμόζεται στο ευρέως χρησιμοποιούμενο λειτουργικό σύστημα UNIX, αλλά βασίζεται μόνο σε μία παραλλαγή του αλγορίθμου DES, ο οποίος δεν χαρακτηρίζεται σαν μία αυστηρά μονόδρομη συνάρτηση.

2.2.3 Μονόδρομες Συναρτήσεις Κρυφής Εισόδου

Για την ασφάλεια δεδομένων, που είναι και ο τομέας της κρυπτογραφίας, οι μονόδρομες συναρτήσεις κρυφής εισόδου είναι απαραίτητες αφού επιτρέπουν την πρόσβαση δεδομένων μέσω της αποκρυπτογράφησης από ένα εξουσιοδοτημένο χρήστη. Μία αμφιμονοσήμαντη συνάρτηση $f : X \to Y$ ονομάζεται μονόδρομη συνάρτηση κρυφής εισόδου αν ισχύουν τα ακόλουθα:

1. Υπάρχει μία αποδοτική μέθοδος υπολογισμού της $f(x)$ για όλα τα $x \in X$.
2. Υπάρχει μία αποδοτική μέθοδος υπολογισμού της $f^{-1}(y)$ για όλα τα $y \in f[X]$, αλλά δε μπορεί να παραχθεί αποδοτικά από τη σχέση $y = f(x)$ για όλα τα $y \in f[X]$ καθώς είναι απαραίτητη μία επιπλέον μυστική πληροφορία, η «καταπακτή» (*trapdoor*).

2.2.4 Ένα Παράδειγμα Μονόδρομης Συνάρτησης

Η απόδειξη για την ύπαρξη αυστηρά μονόδρομων συναρτήσεων παρεμποδίζεται από την έλλειψη αποδοτικά καλών χαμηλών ορίων στις γνωστές μεθόδους. Αλλά υπάρχουν καλοί υποψήφιοι που βασίζονται σε πράξεις πολλαπλασιασμού και εκθετικοποίησης στο σώμα Galois F(*p*), όπου το *p* είναι πρώτος.

Παράδειγμα 10. Μονόδρομη συνάρτηση χωρίς κρυφή πληροφορία: Πολλαπλασιασμός πρώτων αριθμών.

Είναι σχετικά απλό να πολλαπλασιάσεις δύο αριθμούς με δέκα χιλιάδες δεκαδικά ψηφία ο καθένας και άρα το ίδιο εύκολο είναι να το κάνεις για δύο πρώτους, αφού ακόμα και σε ένα προσωπικό υπολογιστή θα διαρκέσει μόνο κάποια δευτερόλεπτα. Αλλά σήμερα δεν υπάρχει αποδοτική μέθοδος (δημόσια γνωστή) που να αποσυνθέτει ένα αριθμό 200 ψηφίων στους πρωταρχικούς του παράγοντες (εκτός από ειδικές περιπτώσεις).

Ας υποθέσουμε $X = \{(x_1, x_2) \mid x_1, x_2 \ \textit{πρώτοι}, K \le x_1 \le x_2\}$ για ένα αρκετά μεγάλο *K*. Η αμφιμονοσήμαντη συνάρτηση $f : X \to \mathbb{N}$ που ορίζεται ως $f(x_1, x_2) = x_1 \cdot x_2$ είναι μονόδρομη (one-way) [18].

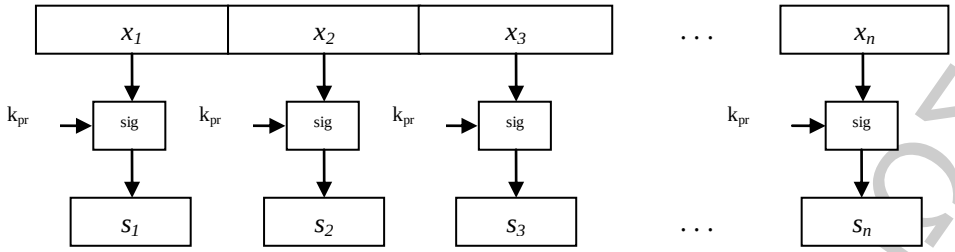
2.3 Συναρτήσεις κατακερματισμού

Οι συναρτήσεις κατακερματισμού (hash functions) αποτελούν σημαντικό θεμελιακό κομμάτι της κρυπτογραφίας και χρησιμοποιούνται ευρέως σε όλα τα πρωτόκολλα. Αυτές υπολογίζουν μία σύνοψη ενός μηνύματος το οποίο είναι μία σύντομη συμβολοσειρά από bit ορισμένου μήκους. Για ένα συγκεκριμένο μήνυμα, η σύνοψη του ή αλλιώς η κατακερματισμένη τιμή του (hash value), μπορεί να θεωρηθεί το αποτύπωμα του μηνύματος, δηλαδή μία μοναδική αναπαράσταση του μηνύματος. Σε αντίθεση πολλών άλλων αλγορίθμων κρυπτογραφίας, οι συναρτήσεις κατακερματισμού δεν έχουν κάποιο κλειδί. Η χρήσης τους στη κρυπτογραφία είναι πολλαπλή καθώς αποτελούν ένα απαραίτητο τμήμα των συστημάτων ψηφιακής υπογραφής και των κωδικών πιστοποίησης μηνύματος. Επίσης χρησιμοποιούνται ευρέως και για άλλες εφαρμογές κρυπτογραφίας, όπως π.χ. για την αποθήκευση κατακερματισμένων κωδικών πρόσβασης και την παραγωγή κλειδιού.

2.3.1 Κίνηρο:Υπογραφή Μεγάλων Μηνυμάτων

Ακόμα κι αν οι συναρτήσεις κατακερματισμού έχουν πολλές εφαρμογές στη σύγχρονη κρυπτογραφία, ίσως είναι πιο γνωστές για το σημαντικό ρόλο που διαδραματίζουν στην πρακτική εφαρμογή των ψηφιακών υπογραφών. Για όλα τα συστήματα, το μήκος του αρχικού μηνύματος είναι περιορισμένο. Για παράδειγμα, στη περίπτωση του RSA, το μήνυμα δε μπορεί να είναι μεγαλύτερο από το συντελεστή, ο οποίος στη πράξη είναι συνήθως μήκους μεταξύ 1024 και 3072 bit, δηλαδή μόνο 128 μέχρι 384 bytes. Ως εδώ είχαμε αγνοήσει το γεγονός ότι το αρχικό μήνυμα θα είναι συχνά πολύ μεγαλύτερο από αυτά τα μεγέθη. Το ερώτημα που προκύπτει σε αυτό το σημείο είναι απλό: Πώς θα υπολογίσουμε αποδοτικά ψηφιακές

υπογραφές μεγάλων μηνυμάτων; Μία διαισθητική προσέγγιση μπορεί να είναι η εξής: Διαιρούμε το μήνυμα *x* σε τμήματα μεγέθους x_i μικρότερα από το επιτρεπόμενο μέγεθος εισόδου του αλγορίθμου υπογραφής και υπογράφουμε κάθε τμήμα χωριστά όπως φαίνεται στην παρακάτω εικόνα.



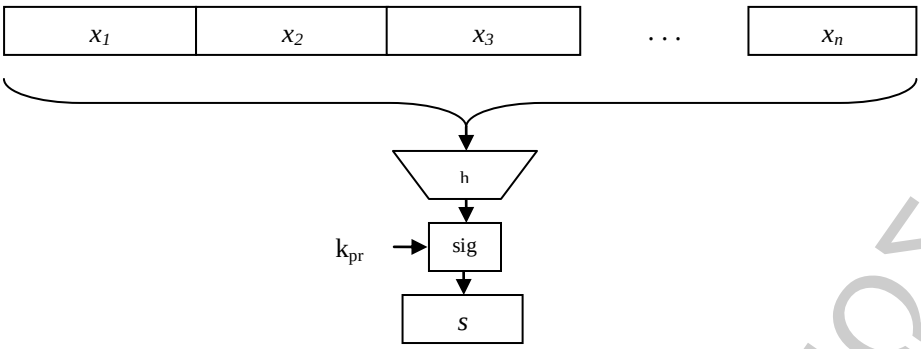
Εικόνα 2. Ανασφαλής προσέγγιση υπογραφής μεγάλων μηνυμάτων[31].

Ωστόσο η προσέγγιση αυτή δημιουργεί τρία σοβαρά προβλήματα:

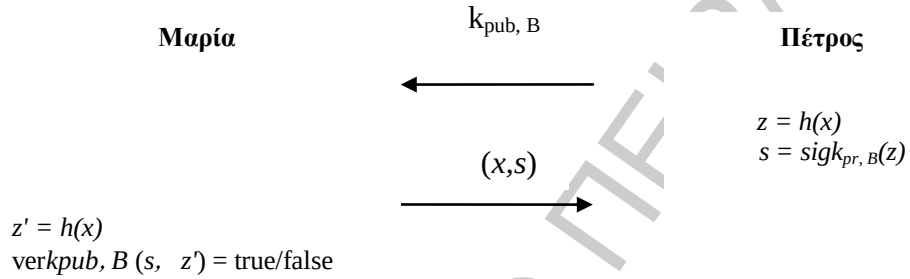
1. *Υψηλό υπολογιστικό φορτίο*: Οι ψηφιακές υπογραφές βασίζονται στις υπολογιστικά έντονες ασύμμετρες πράξεις όπως οι εκθετικοποιήσεις κατά modulo μεγάλων ακεραίων. Ακόμη κι αν μία απλή πράξη καταναλώνει μικρό χρόνο (και ενέργεια), οι υπογραφές μεγάλων μηνυμάτων, π.χ., επισυναπτόμενα αρχεία ηλεκτρονικών μηνυμάτων ή αρχεία πολυμέσων, θα έπαιρνε πολύ χρόνο ακόμα και στους σύγχρονους υπολογιστές. Επιπλέον δεν πρέπει μόνο να υπολογίσει ο υπογράφων την ψηφιακή υπογραφή, αλλά θα πρέπει επίσης και ο επαληθευτής να δαπανήσει τον ίδιο χρόνο και ενέργεια για να επιβεβαιώσει την υπογραφή.
2. *Ωφέλιμο μηνύματος*: Προφανώς αυτή η απλοϊκή προσέγγιση διπλασιάζει το ωφέλιμο του μηνύματος επειδή δεν στέλνουμε μόνο του το μήνυμα αλλά στέλνουμε και την υπογραφή μαζί, η οποία σε αυτή τη περίπτωση έχει το μέγεθος του μηνύματος. Έτσι για παράδειγμα, ένα αρχείο μεγέθους 1 MB θα παράγει μία υπογραφή RSA μήκους 1 MB, που σημαίνει ότι το συνολικό μέγεθος που θα πρέπει να σταλεί θα είναι 2 MB.
3. *Περιορισμοί ασφάλειας*: Αυτό είναι το πιο σημαντικό πρόβλημα που προκύπτει αν προσπαθήσουμε να υπογράψουμε ένα μεγάλο μήνυμα υπογράφοντας μεμονωμένα μία ακολουθία τμημάτων του. Η προσέγγιση που φαίνεται στη παραπάνω εικόνα οδηγεί αμέσως σε νέες επιθέσεις: Κάποιος για παράδειγμα θα μπορούσε να αφαιρέσει μεμονωμένα μηνύματα και τις αντίστοιχες υπογραφές τους, ή να αναδιατάξει τα μηνύματα και τις υπογραφές, ή θα μπορούσε να ξαναδημιουργήσει νέα μηνύματα και υπογραφές πέρα από τα τμήματα των προηγούμενων μηνυμάτων και υπογραφών, κτλ. Ακόμη κι αν ένας επιτιθέμενος δεν μπορεί να εκτελέσει κάποιες ενέργειες μέσα σε ένα μεμονωμένο τμήμα, εμείς δε θα έχουμε προστασία για ολόκληρο το μήνυμα.

Ως εκ τούτου, για την απόδοση καθώς και για λόγους ασφαλείας θα θέλαμε να έχουμε μία σύντομη υπογραφή για ένα μήνυμα αυθαίρετου μεγέθους. Η λύση σε αυτό το πρόβλημα είναι οι συναρτήσεις κατακερματισμού. Αν είχαμε μία τέτοια συνάρτηση η οποία θα υπολόγιζε κάπως ένα αποτύπωμα του μηνύματος *x*, θα μπορούσαμε να εκτελέσουμε την διαδικασία της υπογραφής όπως φαίνεται στη παρακάτω εικόνα.

Ας υποθέσουμε ότι έχουμε μία τέτοια συνάρτηση κατακερματισμού. Θα περιγράψουμε ένα βασικό πρωτόκολλο για ένα σύστημα ψηφιακής υπογραφής με μία συνάρτηση κατακερματισμού. Στο παράδειγμα αυτό ο Πέτρος θέλει να στείλει ένα ψηφιακά υπογεγραμμένο μήνυμα στη Μαρία.



Εικόνα 3. Υπογράφοντας μεγάλα μηνύματα με μία συνάρτηση κατακερματισμού[31].



Εικόνα 4. Βασικό πρωτόκολλο ψηφιακής υπογραφής με συνάρτηση κατακερματισμού[31].

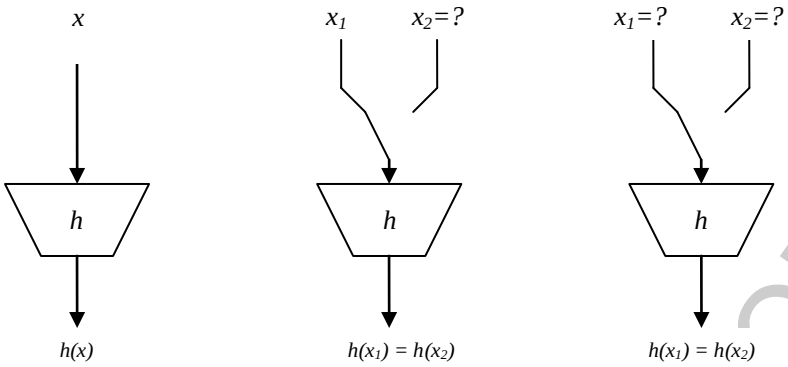
Ο Πέτρος υπολογίζει τη κατακερματισμένη τιμή του μηνύματος *x* και υπογράφει τη κατακερματισμένη τιμή *z* με το ιδιωτικό κλειδί *kpr,B*. Από τη μεριά του παραλήπτη, η Μαρία υπολογίζει τη κατακερματισμένη τιμή *z'* του μηνύματος *x* που παρέλαβε και επαληθεύει την υπογραφή *s* με το δημόσιο κλειδί *kpub,B* του Πέτρου. Σημειώνουμε ότι και η παραγωγή της υπογραφής και η επαλήθευση λειτουργούν επί της κατακερματισμένης τιμής *z* αντί του ίδιου του μηνύματος. Έτσι η κατακερματισμένη τιμή αντιπροσωπεύει το μήνυμα. Ο κατακερματισμός αναφέρεται συχνά ως σύνοψη μηνύματος ή ως αποτύπωμα του μηνύματος (fingerprint of a message) [17].

2.3.2 Απαιτήσεις Ασφαλείας των Συναρτήσεων Κατακερματισμού

Σε αντίθεση με τους άλλους αλγόριθμους κρυπτογραφίας, οι συναρτήσεις κατακερματισμού δεν έχουν κλειδιά. Το ερώτημα που προκύπτει είναι εάν υπάρχουν τυχόν κάποιες ειδικές ιδιότητες που απαιτούνται για μια συνάρτηση κατακερματισμού έτσι ώστε να είναι «ασφαλής». Όντως θα πρέπει να αναρωτηθούμε αν οι συναρτήσεις κατακερματισμού έχουν καθόλου επίπτωση στην ασφάλεια μιας εφαρμογής, αφού, ούτε κρυπτογραφούν, ούτε έχουν κλειδιά. Όπως συμβαίνει συχνά στην κρυπτογραφία, τα πράγματα μπορεί να είναι περίπλοκα και υπάρχουν επιθέσεις οι οποίες χρησιμοποιούν τις αδυναμίες των συναρτήσεων κατακερματισμού. Αποδεικνύεται ότι υπάρχουν τρεις βασικές ιδιότητες τις οποίες θα πρέπει να κατέχουν οι συναρτήσεις αυτές για να είναι ασφαλείς:

1. Μονόδρομες
2. Ασθενής αντίσταση σε συγκρούσεις
3. Ισχυρή αντίσταση σε συγκρούσεις

Τις τρεις αυτές ιδιότητες οπτικοποιούμε στη παρακάτω εικόνα.



Εικόνα 5. Οι τρεις ιδιότητες ασφαλείας των συναρτήσεων κατακερματισμού[31].

Η συνάρτηση κατακερματισμού πρέπει να είναι one-way. Δηλαδή δοθέντος μίας κατακερματισμένης εξόδου *z*, πρέπει να είναι υπολογιστικά ανέφικτο να βρεθεί ένα μήνυμα εισόδου *x* που να ισχύει $z = h(x)$. Με άλλα λόγια, δοθέντος ενός ψηφιακού αποτυπώματος, δε μπορούμε να παράγουμε το αρχικό μήνυμα. Θα επιδείξουμε τώρα γιατί αυτή η ιδιότητα είναι σημαντική με την έννοια ενός πλασματικού πρωτοκόλλου κατά το οποίο ο Πέτρος κρυπτογραφεί το μήνυμα αλλά όχι την υπογραφή, δηλαδή, διαβιβάζει το ζεύγος:

$$(e_k(x), \text{sig}_{kpr,B}(z)).$$

Εδώ, το $e_k(x)$ είναι ένα συμμετρικός κρυπταλγόριθμος, π.χ. ο AES, με κάποια συμμετρικά κλειδιά που έχουν μοιραστεί από τον Πέτρο και τη Μαρία. Ας υποθέσουμε ότι ο Πέτρος χρησιμοποιεί μία ψηφιακή υπογραφή RSA, όπου αυτή υπολογίστηκε όπως παρακάτω:

$$s = \text{sig}_{kpr,B}(z) \equiv z^d \bmod n$$

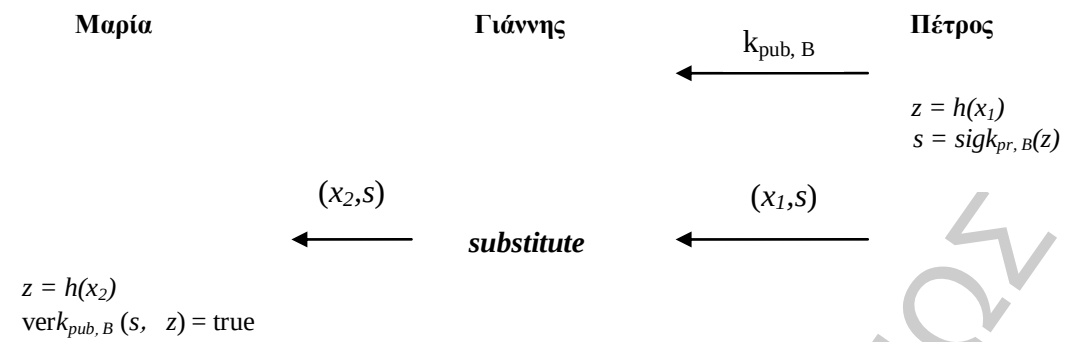
Ο επίδοξος επιτιθέμενος μπορεί να χρησιμοποιήσει το δημόσιο κλειδί του Πέτρου για να υπολογίσει το: $s^e \equiv z \bmod n$.

Αν η συνάρτηση κατακερματισμού δεν είναι μονόδρομη (one-way), ο επιτιθέμενος μπορεί τώρα να υπολογίσει το μήνυμα *x* ως εξής, $h^{-1}(z) = x$. Έτσι, η συμμετρική κρυπτογράφηση του *x* καταστρατηγείται από την υπογραφή, η οποία διαρρέει το αρχικό μήνυμα. Για αυτό το λόγο, η *h(x)* πρέπει να είναι μονόδρομη συνάρτηση. Σε πολλές άλλες εφαρμογές που χρησιμοποιούν συναρτήσεις κατακερματισμού, για παράδειγμα στη παραγωγή κλειδιού, το γεγονός ότι είναι preimage resistant είναι ακόμη πιο σημαντικό[16,30,31,32].

2.3.3 Ασθενής αντίσταση σε συγκρούσεις

Για ψηφιακές υπογραφές που παράγονται με κατακερματισμό, είναι απαραίτητο δύο διαφορετικά αρχικά μηνύματα να μην κατακερματίζονται στην ίδια τιμή. Αυτό σημαίνει ότι θα πρέπει να είναι υπολογιστικά ανέφικτο να δημιουργηθούν δύο διαφορετικά μηνύματα $x_1 \neq x_2$ με ίδιες τιμές κατακερματισμού $z_1 = h(x_1) = h(x_2) = z_2$. Κάνουμε διάκριση μεταξύ δύο διαφορετικών τύπων στις εν λόγω συγκρούσεις. Στη πρώτη περίπτωση, δίνεται το x_1 και προσπαθούμε να βρούμε το x_2. Αυτό ονομάζεται second preimage resistance. Η δεύτερη περίπτωση συμβαίνει εάν ένας επιτιθέμενος είναι ελεύθερος να διαλέξει το x_1 και το x_2. Αυτό αναφέρεται ως strong collision resistance και θα το δούμε παρακάτω.

Είναι εύκολο να δούμε γιατί η second preimage resistance είναι σημαντική για το βασικό σύστημα υπογραφής με κατακερματισμό που περιγράψαμε παραπάνω. Υποθέτουμε ότι ο Πέτρος κατακερματίζει και υπογράφει ένα μήνυμα x_1. Αν ο επιτιθέμενος, π.χ. ο Γιάννης, είναι σε θέση να βρει ένα δεύτερο μήνυμα x_2 που να ισχύει $h(x_1) = h(x_2)$, μπορεί τότε να εκτελέσει την ακόλουθη επίθεση αντικατάστασης:



Εικόνα 6. Επίθεση αντικατάστασης[30].

Όπως μπορούμε να δούμε, η Μαρία θα δεχτεί το x_2 σαν ένα σωστό μήνυμα, αφού η επαλήθευση δίνει σε αυτήν την απάντηση «αληθές». Πώς λοιπόν μπορεί να γίνει αυτό; Από μία πιο αφηρημένη οπτική, η επίθεση αυτή είναι δυνατή επειδή και η υπογραφή (από τον Πέτρο) και η επαλήθευση (από τη Μαρία) δεν γίνεται με το ίδιο το πραγματικό μήνυμα, αλλά με τη κατακερματισμένη έκδοση αυτού. Ως εκ τούτου, αν ένας επιτιθέμενος καταφέρει να βρει ένα δεύτερο μήνυμα με το ίδιο ψηφιακό αποτύπωμα (δηλαδή η έξοδος που παράγει η συνάρτηση κατακερματισμού), η υπογραφή και η επαλήθευση είναι η ίδια για αυτό το δεύτερο μήνυμα.

Το ερώτημα που προκύπτει τώρα είναι πως μπορούμε να εμποδίσουμε το Γιάννη να βρει το x_2. Ιδανικά θα θέλαμε να έχουμε μία συνάρτηση κατακερματισμού για την οποία δεν θα υπάρχουν αδύναμες συγκρούσεις. Αυτό δυστυχώς είναι αδύνατον σύμφωνα με την αρχή του περιστερώνα (pigeonhole principle). Η αρχή αυτή χρησιμοποιεί ένα επιχείρημα μέτρησης σε καταστάσεις όπως η ακόλουθη: Αν είσαι ιδιοκτήτης 100 περιστεριών, αλλά στον περιστερώνα σου υπάρχουν μόνο 99 φωλιές, τότε τουλάχιστον μία φωλιά θα καταλαμβάνετε από 2 πουλιά. Αφού η έξοδος κάθε συνάρτησης κατακερματισμού έχει ένα καθορισμένο μήκος bit, ας υποθέσουμε n bit, υπάρχουν μόνο 2^n πιθανές τιμές εξόδου. Την ίδια στιγμή, το πλήθος των εισόδων στις συναρτήσεις κατακερματισμού είναι άπειρο, έτσι πολλαπλές είσοδοι θα πρέπει να κατακερματιστούν στην ίδια τιμή εξόδου. Στη πράξη, κάθε τιμή εξόδου είναι εξίσου πιθανή για μία τυχαία είσοδο, έτσι η αδύναμη αυτή σύγκρουση υπάρχει για όλες τις τιμές εξόδου.

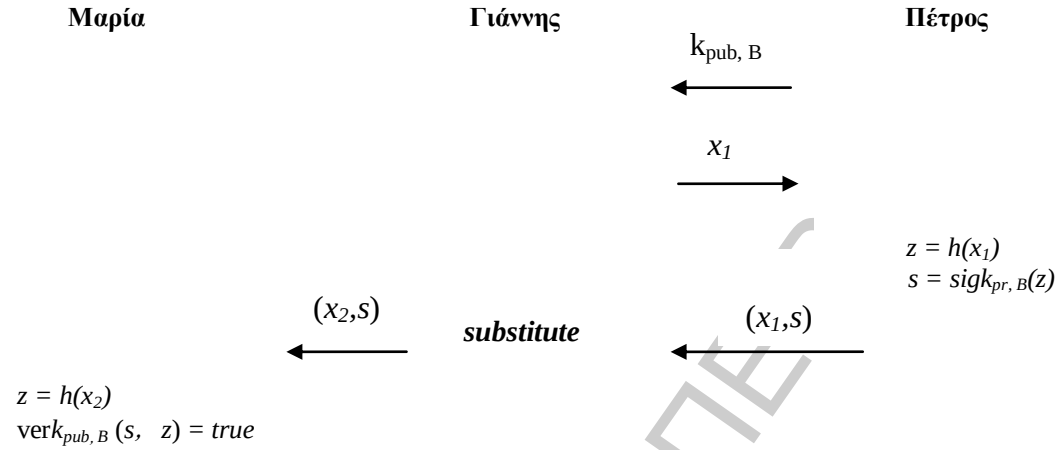
Αφού λοιπόν υπάρχουν αδύναμες συγκρούσεις στη θεωρία, το επόμενο πράγμα που μπορούμε να κάνουμε είναι να διασφαλίσουμε ότι δε θα μπορούν να βρεθούν στη πράξη. Μία ισχυρή συνάρτηση κατακερματισμού θα πρέπει να σχεδιαστεί έτσι ώστε, δοθέντων x_1 και $h(x_1)$ να είναι αδύνατο να κατασκευαστεί x_2 τέτοιο ώστε να ισχύει $h(x_1) = h(x_2)$. Αυτό σημαίνει ότι δεν υπάρχει καμία αναλυτική επίθεση. Ωστόσο, ο Γιάννης μπορεί πάντα να συλλέγει x_2 τιμές, να υπολογίζει τις κατακερματισμένες τιμές και να ελέγχει κάθε φορά πότε είναι ίσες με αυτές της $h(x_1)$. Αυτή η διαδικασία είναι παρόμοια με την εξαντλητική αναζήτηση κλειδιού για ένα συμμετρικό κρυπταλγόριθμο. Για να εμποδίσουμε αυτή την επίθεση, με τους δεδομένους υπολογιστές της εποχής, μία έξοδο μήκους $n = 80$ bit είναι επαρκής. Ωστόσο θα δούμε παρακάτω ότι υπάρχουν πιο ισχυρές επιθέσεις που μας αναγκάζουν να χρησιμοποιούμε ακόμη περισσότερα bit στο μήκος των εξόδων[16,30,31,32].

2.3.4 Ισχυρή αντίσταση σε συγκρούσεις

Αποκαλούμε μία συνάρτηση κατακερματισμού ανθεκτική στη σύγκρουση αν είναι υπολογιστικά ανέφικτο να βρεθούν δύο διαφορετικές είσοδοι $x_1 \neq x_2$ με $h(x1) = h(x2)$. Αυτή η ιδιότητα είναι πιο δύσκολο να επιτευχθεί από την weak collision resistance αφού ο επιτιθέμενος έχει δύο βαθμούς ελευθερίας: Αμφότερα τα μηνύματα μπορούν να μεταβάλλονται για να επιτευχθούν όμοιες τιμές κατακερματισμού (hash values). Θα δείξουμε τώρα πώς ο Γιάννης θα μπορούσε να μετατρέψει την ικανότητα του για να βρει συγκρούσεις σε μία επίθεση. Ξεκινάει με δύο μηνύματα, για παράδειγμα[30]:

x_1 = *Μετέφερε 10€ στο λογαριασμό του Γιάννη*
x_2 = *Μετέφερε 10.000€ στο λογαριασμό του Γιάννη*

Αυτός τώρα μεταβάλει τα x_1 *και* x_2 σε «μη ορατές» θέσεις, για παράδειγμα, αντικαθιστά τα διαστήματα με tabs, προσθέτει διαστήματα, κτλ. Με αυτό τον τρόπο η σημασιολογία του μηνύματος παραμένει αμετάβλητη (π.χ. για μία τράπεζα), αλλά η κατακερματισμένη τιμή αλλάζει για κάθε εκδοχή του μηνύματος. Ο Γιάννης συνεχίζει, μέχρι να πληρείται η συνθήκη *h(x1) = h(x2)*. Σημειώστε ότι αν ένας επιτιθέμενος έχει για παράδειγμα 64 θέσεις που μπορεί να αλλάξει ή όχι, αυτό παράγει 2^{64} εκδοχές του ίδιου μηνύματος με 2^{64} διαφορετικές κατακερματισμένες τιμές. Με τα δύο μηνύματα μπορεί να προωθήσει την ακόλουθη επίθεση:



Εικόνα 7. Επίθεση αντικατάστασης δύο μηνυμάτων[30].

Η επίθεση αυτή υποθέτει ότι ο Γιάννης μπορεί να ξεγελάσει τον Πέτρο στο να υπογράψει το μήνυμα x_1. Αυτό φυσικά, δεν είναι δυνατό σε κάθε περίπτωση. Όπως είδαμε νωρίτερα, σύμφωνα με την αρχή του περιστερώνα, συγκρούσεις πάντα υπάρχουν. Το ερώτημα είναι πόσο δύσκολο είναι να τις βρεις. Η πρώτη μας εικασία είναι ότι ίσως αυτό είναι τόσο δύσκολο όσο να βρεις second preimages, δηλαδή, αν η συνάρτηση κατακερματισμού έχει μία έξοδο μήκους 80 bit, θα πρέπει εμείς να ελέγξουμε 2^{80} μηνύματα. Ωστόσο αποδεικνύεται ότι ένας επιτιθέμενος χρειάζεται μόνο 2^{40} μηνύματα να ελέγξει. Αυτό είναι αρκετά εκπληκτικό αποτέλεσμα που οφείλεται στην επίθεση γενεθλίων (birthday attack). Η επίθεση αυτή βασίζεται στο παράδοξο των γενεθλίων (birthday paradox), το οποίο είναι ένα ισχυρό εργαλείο που συχνά χρησιμοποιείται στην κρυπτανάλυση.

Συνοψίζοντας τα παραπάνω, θα λέγαμε ότι οι συναρτήσεις κατακερματισμού χαρακτηρίζονται από τις παρακάτω έξι ιδιότητες:

1. *Αυθαίρετο μέγεθος μηνύματος*: Η *h*(*x*) μπορεί να εφαρμοστεί σε μηνύματα *x* οποιουδήποτε μεγέθους.
2. *Καθορισμένο μήκος εξόδου*: Η *h*(*x*) παράγει μία κατακερματισμένη τιμή *z* καθορισμένου μήκους.
3. *Αποδοτικότητα*: Η *h*(*x*) είναι σχετικά εύκολο να υπολογιστεί.
4. *Preimage resistance*: Για μία δοθέν έξοδο *z*, είναι αδύνατο να βρεθεί είσοδος *x* τέτοια ώστε *h(x) = z*, δηλαδή, η *h(x)* είναι one-way.
5. *Second preimage resistance*: Δοθέντος x_1, άρα και $h(x_1)$, είναι υπολογιστικά ανέφικτο να βρεθεί οποιοδήποτε x_2 τέτοιο ώστε $h(x_1) = h(x_2)$.
6. *Collision resistance*: Είναι υπολογιστικά αδύνατο να βρεθεί ζεύγος $x_1 = x_2$ τέτοιο ώστε $h(x_1) = h(x_2)$.

2.4 Το πρόβλημα του διακριτού λογαρίθμου

Στο πρώτο Κεφάλαιο της παρούσας εργασίας αναφερθήκαμε και παρουσιάσαμε ορισμούς και προτάσεις της αλγοριθμικής άλγεβρας. Κάποια από αυτά αφορούσαν τις κυκλικές ομάδες. Αποδεικνύεται ότι η μονόδρομη συνάρτηση DLP (Discrete Logarithm Problem – Πρόβλημα Διακριτού Λογαρίθμου) του πρωτοκόλλου DHKE (Diffie Hellman Key Exchange) μπορεί να επεξηγηθεί άμεσα χρησιμοποιώντας κυκλικές ομάδες.

2.4.1 Το Πρόβλημα του Διακριτού Λογαρίθμου σε Πρωταρχικά Σώματα

Ξεκινάμε με το DLP πάνω στο $\mathbb{Z}_p^*$, όπου το *p* είναι πρώτος.

Ορισμός 15. *Έστω ότι έχουμε μία πεπερασμένη κυκλική ομάδα* $\mathbb{Z}_p^*$ *τάξης p-1, ένα πρωταρχικό στοιχείο* $a \in \mathbb{Z}_p^*$ *και ένα άλλο στοιχείο* $\beta \in \mathbb{Z}_p^*$. *Το DLP αποτελεί το πρόβλημα καθορισμού – εύρεσης του ακεραίου* $1 \le x \le p-1$ *για τον οποίο ισχύει:* $a^x \equiv \beta \bmod p$.

Ένας τέτοιος ακέραιος *x* πρέπει να υπάρχει αφού το *a* είναι ένα πρωταρχικό στοιχείο και κάθε ομαδικό στοιχείο μπορεί να εκφραστεί σαν μια δύναμη οποιουδήποτε πρωταρχικού στοιχείου. Ο ακέραιος αυτός *x* ονομάζεται διακριτός λογάριθμος του *β* στη βάση του *a*, και μπορούμε να γράψουμε τυπικά ότι: $x = \log_a \beta \bmod p$.

Ο υπολογισμός διακριτών λογαρίθμων κατά modulo ενός πρώτου αριθμού είναι ένα πάρα πολύ δύσκολο πρόβλημα αν οι παράμετροι είναι αρκετά μεγάλοι. Επειδή η εκθετικοποίηση $a^x \equiv \beta \bmod p$ είναι υπολογιστικά εύκολη, αυτό σχηματίζει μία μονόδρομη συνάρτηση.

Παράδειγμα 11. Θεωρούμε ένα διακριτό λογάριθμο στην ομάδα $\mathbb{Z}_{47}^*$, στην οποία το *a* = 5 είναι ένα πρωταρχικό στοιχείο. Για *β* = 41 το πρόβλημα διακριτού λογαρίθμου είναι: Βρες το θετικό ακέραιο *x* τέτοιο ώστε: $5^x \equiv 41 \bmod 47$.

Ακόμη και για τόσους μικρούς αριθμούς, ο καθορισμός του *x* δεν είναι απόλυτα απλή διαδικασία. Στη πράξη είναι συχνά επιθυμητό να έχουμε ένα DLP σε ομάδες με πληθικότητα πρώτου αριθμού για να εμποδίσουμε τυχόν επίθεση τύπου Pohlig – Hellman. Επειδή οι ομάδες $\mathbb{Z}_p^*$ έχουν πληθικότητα ίση με *p – 1*, η οποία προφανώς δεν είναι πρώτος αριθμός, συχνά χρησιμοποιείται DLPs σε υποομάδες του $\mathbb{Z}_p^*$ με πρωταρχική τάξη, αντί να χρησιμοποιείται η ίδια η ομάδα $\mathbb{Z}_p^*$. Στη συνέχεια επεξηγούμε το παραπάνω με ένα παράδειγμα.

Παράδειγμα 12. Έστω η ομάδα $\mathbb{Z}_{47}^*$ η οποία έχει τάξη ίση με 46. Έτσι οι υποομάδες στο $\mathbb{Z}_{47}^*$ έχουν πληθικότητα 23, 2 και 1. Το *a* = 2 είναι ένα στοιχείο στην υποομάδα με 23 στοιχεία, και αφού το 23 είναι πρώτος, το *a* είναι ένα πρωταρχικό στοιχείο στην υποομάδα. Ένα πιθανό πρόβλημα διακριτού λογαρίθμου δίνεται για *β* = 36 (το οποίο βρίσκεται στην υποομάδα): Βρείτε το θετικό ακέραιο *x*, $1 \le x \le 23$ τέτοιο ώστε: $2^x \equiv 36 \bmod 47$. Εκτελώντας μία δυνατή εξαντλητική επίθεση, βρίσκουμε μία λύση για *x* = 17.

2.4.2 Το Γενικευμένο Πρόβλημα Διακριτού Λογαρίθμου

Το χαρακτηριστικό που κάνει το DLP ιδιαίτερα χρήσιμο στην κρυπτογραφία είναι το γεγονός ότι δεν περιορίζεται στην πολλαπλασιαστική ομάδα $\mathbb{Z}_p^*$, όπου *p* πρώτος, αλλά μπορεί να οριστεί σε οποιεσδήποτε κυκλικές ομάδες. Αυτό ονομάζεται *γενικευμένο πρόβλημα διακριτού λογαρίθμου (GDLP)* και μπορεί να οριστεί ως εξής:

Ορισμός 16. *Έστω ότι έχουμε μία πεπερασμένη κυκλική ομάδα G με τη δυαδική πράξη ομάδας* $\circ$ *και πληθικότητα ίση με n. Θεωρούμε ένα πρωταρχικό στοιχείο* $a \in G$ *και ένα άλλο στοιχείο* $\beta \in G$. *Το πρόβλημα διακριτού λογαρίθμου είναι να βρεθεί ο ακέραιος x, όπου* $1 \le x \le n$, *τέτοιο ώστε:*

$$\beta = \underbrace{\alpha \circ \alpha \circ \ldots \circ \alpha}_{x \text{ φορές}} = \alpha^x$$

Όπως στην περίπτωση του DLP στο $\mathbb{Z}_p^*$, ένας τέτοιος ακέραιος *x* πρέπει να υπάρχει, αφού το *a* είναι ένα πρωταρχικό στοιχείο και έτσι κάθε στοιχείο της ομάδας *G* μπορεί να παραχθεί με επαναλαμβανόμενη εφαρμογή της πράξης της ομάδας στο *a*.

Είναι σημαντικό να καταλάβουμε ότι υπάρχουν κυκλικές ομάδες στις οποίες το DLP δεν είναι δύσκολο. Τέτοιες ομάδες δεν μπορούν να χρησιμοποιηθούν για κρυπτοσυστήματα δημοσίου κλειδιού, αφού το DLP δεν είναι μονόδρομη συνάρτηση. Θεωρείστε το ακόλουθο παράδειγμα.

Παράδειγμα 13. Αυτή τη φορά θεωρούμε την προσθετική ομάδα των ακεραίων modulo ένα πρώτο αριθμό. Για παράδειγμα, εάν επιλέξουμε τον πρώτο $p = 11$, η $G = (\mathbb{Z}_{11}, +)$ είναι μία πεπερασμένη κυκλική ομάδα με πρωταρχικό στοιχείο *a* = 2. Εδώ είναι το πώς το *a* παράγει την ομάδα:

i	1	2	3	4	5	6	7	8	9	10	11
i a	2	4	6	8	10	1	3	5	7	9	0

Θα προσπαθήσουμε τώρα να λύσουμε το DLP για το στοιχείο *β* = 3, δηλαδή, πρέπει να υπολογίσουμε τον ακέραιο $1 \le x \le 11$ τέτοιο ώστε να ισχύει

$$x \cdot 2 = \underbrace{2 + 2 + \ldots + 2}_{x \text{ φορές}} \equiv 3 \bmod 11$$

Τώρα θα δείξουμε πώς δουλεύει μία επίθεση ενάντια στο DLP. Παρόλο που η πράξη της ομάδας είναι πρόσθεση, μπορούμε να εκφράσουμε τη σχέση ανάμεσα στα *a* και *β* και στο διακριτό λογάριθμο *x* με όρους του πολλαπλασιασμού: $x \cdot 2 \equiv 3 \bmod 11$. Για να λύσουμε για *x*, πρέπει απλά να αντιστρέψουμε το πρωταρχικό στοιχείο *a*: $x \equiv 2^{-1} 3 \bmod 11.$

Χρησιμοποιώντας, π.χ., τον επεκταμένο αλγόριθμο του Ευκλείδη, μπορούμε να υπολογίσουμε $2^{-1} \equiv 6 \bmod 11$ από την οποία ο διακριτός λογάριθμος έχει ως εξής: $x = 2^{-1}\, 3 \equiv 7 \bmod 11.$

Μπορούμε να γενικεύσουμε το παραπάνω τέχνασμα σε οποιαδήποτε ομάδα $(\mathbb{Z}_n, +)$ για ένα αυθαίρετο *n* και στοιχεία $\alpha, \beta \in \mathbb{Z}_n$. Ως εκ τούτου, συμπεραίνουμε ότι το γενικευμένο DLP είναι υπολογιστικά εύκολο στο $\mathbb{Z}_n$. Ο λόγος που το DLP μπορεί να εύκολα να επιλυθεί εδώ, είναι ότι έχουμε μαθηματικές πράξεις που δεν είναι στη προσθετική ομάδα, όπως, πολλαπλασιασμό και αντιστροφή.

Μετά από αυτό το αντιπαράδειγμα, παραθέτουμε μία λίστα με τα προβλήματα διακριτού λογαρίθμου που έχουν προταθεί για χρήση στην κρυπτογραφία.

1. Η πολλαπλασιαστική ομάδα του πρωταρχικού σώματος $\mathbb{Z}_p$ ή μία υποομάδα του. Για παράδειγμα, ο κλασικός αλγόριθμος DHKE χρησιμοποιεί αυτήν την ομάδα, όπως επίσης η κρυπτογράφηση Elgamal και ο αλγόριθμος ψηφιακής υπογραφής DSA (Digital Signature Algorithm). Αυτοί είναι οι παλαιότεροι και οι πιο ευρέως χρησιμοποιούμενοι τύποι των συστημάτων διακριτού λογαρίθμου.
2. Η κυκλική ομάδα σχηματίζετε από μία ελλειπτική καμπύλη. Τα κρυπτοσυστήματα ελλειπτικών καμπυλών θα τα δούμε παρακάτω. Έχουν γίνει αρκετά δημοφιλή κατά τη τελευταία δεκαετία.
3. Η πολλαπλασιαστική ομάδα του σώματος Galois GF(2^m) ή μία υποομάδα του. Οι ομάδες αυτές μπορούν να χρησιμοποιηθούν απολύτως ανάλογα με τις πολλαπλασιαστικές ομάδες των πρωταρχικών σωμάτων, και σχήματα όπως το DHKE μπορούν να υλοποιηθούν με αυτά. Δεν είναι τόσο δημοφιλή στη πραγματικότητα εξαιτίας του γεγονότος ότι οι επιθέσεις που μπορούν να γίνουν εναντίον τους είναι πιο ισχυρές από αυτές που μπορούν να γίνουν ενάντια στο DLP στο $\mathbb{Z}_p$. Ως εκ τούτου το DLP στο GF(2^m) απαιτεί αρκετά περισσότερα μήκη bit για να παρέχει το ίδιο επίπεδο ασφάλειας με αυτά στο $\mathbb{Z}_p$.
4. Υπέρ – Ελλειπτικές καμπύλες, οι οποίες μπορούν να θεωρηθούν ως γενίκευση των ελλειπτικών καμπυλών. Αυτές σήμερα σπανίως χρησιμοποιούνται στη πράξη, αλλά ειδικότερα οι υπέρ – ελλειπτικές καμπύλες έχουν κάποια πλεονεκτήματα, όπως τα μικρά μήκη τελεστών.

2.5 Το πρόβλημα της παραγοντοποίησης

Ένας ακέραιος αριθμός $n > 1$ λέγεται ότι είναι πρώτος αν οι μοναδικοί διαιρέτες του είναι ο εαυτός του και το 1. Υπάρχουν απείρως πολλοί άπειροι πρώτοι αριθμοί, με τους τέσσερις πρώτους να είναι οι: 2, 3, 5 και 7. Αν ισχύει $n > 1$ και ο n δεν είναι πρώτος, τότε λέγεται ότι είναι σύνθετος. Ο ακέραιος 1 δεν είναι ούτε πρώτος ούτε σύνθετος. Να σημειώσουμε ότι το μαθηματικό υπόβαθρο που απαιτείται για να γίνει εύκολα κατανοητό το πρόβλημα της παραγοντοποίησης βρίσκεται στις ενότητες 2.2.1 και 2.2.2.

Το Θεμελιώδες Θεώρημα της Αριθμητικής δηλώνει ότι κάθε θετικός ακέραιος μπορεί να εκφραστεί σαν ένα πεπερασμένο γινόμενο πρώτων αριθμών και ότι η παραγοντοποίηση αυτή είναι μοναδική, με εξαίρεση τη τάξη των παραγόντων.

Για παράδειγμα έχουμε: $1990 = 2 \cdot 5 \cdot 199$ ή $2000 = 2^4 \cdot 5^3$ ή $2008 = 2^3 \cdot 251$. Η ύπαρξη της παραγοντοποίησης των προηγούμενων παραδειγμάτων είναι ένα εύκολο επακόλουθο του ορισμού του πρώτου αριθμού και της αρχής του καλά διατασσόμενου συνόλου. Η απόδειξη όμως της μοναδικότητας είναι ελαφρώς δυσκολότερη. Ωστόσο αυτή η απόδειξη ύπαρξης δε δίνει καμία ένδειξη πώς να βρίσκει κανείς αποδοτικά τους παράγοντες ενός δοθέντος μεγάλου ακέραιου. Σήμερα δεν υπάρχει γνωστός αλγόριθμος πολυωνυμικού χρόνου που να λύνει αυτό το πρόβλημα.

Η παραγοντοποίηση αποτελούσε κάποτε κατά κύριο λόγο ένα ακαδημαϊκό ενδιαφέρον και απέκτησε πρακτική σημασία μετά την εισαγωγή του κρυπτοσυστήματος δημοσίου κλειδιού RSA (Ενότητα 3.2), η κρυπτογραφική «δύναμη» του οποίου εξαρτάται από τη δυσκολία της παραγοντοποίησης μεγάλων αριθμών.

Ας υποθέσουμε ότι ο N είναι ένας μεγάλος σύνθετος ακέραιος. Μέχρι τη δεκαετία του 1960, οι καλύτεροι αλγόριθμοι παραγοντοποίησης του N κατανάλωναν χρόνο της τάξης $O(N^e)$ για $e > 0$. Ένας τέτοιος αλγόριθμος είναι ο *trial division*, ο οποίος προσπαθεί να διαιρέσει τον N με όλους τους πρώτους μέχρι τον $\sqrt{N}$. Αυτό άλλαξε αργότερα όταν οι Morrison και Brillhart εισήγαγαν τη μέθοδο *continued fraction* της οποίας ο χρόνος είναι:

$$\exp\left\{\left(\sqrt{2}+o(1)\right)\left(\log N \log\log N\right)^{1/2}\right\} = O\left(N\sqrt{\left(2+o(1)\right)\log\log N / \log N}\right).$$

Οι σύγχρονοι αλγόριθμοι που χρησιμοποιούνται για τη παραγοντοποίηση του N διαιρούνται σε δύο κύριες κατηγορίες. Στη πρώτη ανήκουν εκείνοι οι αλγόριθμοι που βρίσκουν μικρούς πρώτους παράγοντες αρκετά γρήγορα. Σε αυτούς περιλαμβάνονται οι *trial division*, *Pollard Rho*, $P \pm 1$ και η μέθοδος της ελλειπτικής καμπύλης. Στη δεύτερη κατηγορία ανήκουν εκείνοι που παραγοντοποιούν έναν αριθμό ανεξαρτήτως του μεγέθους των πρώτων παραγόντων, αλλά καταναλώνουν πολύ περισσότερο χρόνο όταν εφαρμόζονται σε μεγαλύτερους αριθμούς. Σε αυτούς περιλαμβάνονται οι *continued fraction*, *quadratic sieve* και ο *number field sieve*. Να σημειώσουμε ότι στη πράξη, οι αλγόριθμοι και στις δύο κατηγορίες είναι εξίσου σημαντικοί.

Όπως αναφέραμε το πρόβλημα της παραγοντοποίησης μεγάλων αριθμών βρήκε πρακτικό ενδιαφέρον με την ανάπτυξη του RSA, οποίος απαιτεί κάθε χρήστης να έχει τη δική του διαδικασία κρυπτογράφησης E και τη δική του μυστική διαδικασία αποκρυπτογράφησης D. Για κάθε χρήστη η διαδικασία E είναι γνωστή σε όλους τους υπόλοιπους χρήστες. Μόνο όμως ο κάθε χρήστης γνωρίζει τη μυστική του D. Οι διαδικασίες αυτές πρέπει να είναι αμφιμονοσήμαντες και αντίστροφες μεταξύ τους $D(E(M)) = E(D(M)) = M$, για όλα τα μηνύματα M. Και η E και η D πρέπει να είναι γρήγορες στην εκτέλεση τους και υπολογιστικά ακατόρθωτο να βρεθεί η D δοθέντος του E.

Ο RSA πετυχαίνει αυτούς τους στόχους επιτρέποντας σε κάθε χρήστη να διαλέξει δύο μεγάλους πρώτους αριθμούς, τους p και q, με $p \neq q$. Ας υποθέσουμε $N = p \cdot q$, επιλέγουμε δύο εκθέτες e, d με $d \cdot e \equiv 1 \pmod{(p-1)(q-1)}$. Εάν ο M είναι ένας ακέραιος, τότε σύμφωνα με το μικρό θεώρημα του *Fermat* αυτό σημαίνει ότι $M^{de} \equiv M \pmod{p}$ και $M^{de} \equiv M \pmod{q}$. Έτσι $M^{de} \equiv M \pmod{N}$ για όλα τα M. Δηλώνουμε τις διαδικασίες κρυπτογράφησης και αποκρυπτογράφησης να είναι η $E(M) \equiv M^e \pmod{N}$ και $D(M) \equiv M^d \pmod{N}$ αντίστοιχα. Οι τιμές των e και N είναι δημόσιες, ενώ των d, p, q είναι μυστικές.

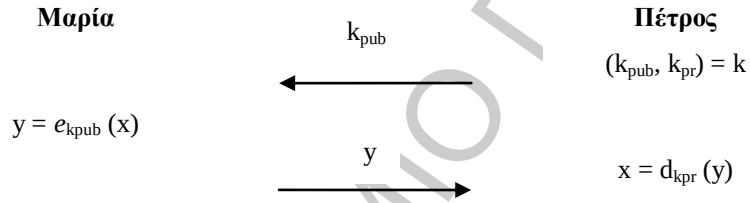
Τα παραπάνω πληρούν τις προϋποθέσεις του δημόσιου κλειδιού εκτός ίσως από την απαίτηση ότι είναι δύσκολο να βρεθεί το D από το E. Εάν η παραγοντοποίηση είναι εύκολη, τότε ένας παρείσακτος μπορεί να βρει τα p και q δοθέντος του $N = p \cdot q$, και από αυτά μπορεί να βρει

το *(p - 1)(q - 1)* και από εκεί το *d* αφού γνωρίζει το *e*. Αυτό σημαίνει ότι είναι εύκολο να βρεις το *D* δοθέντος του *E* αν η παραγοντοποίηση είναι εύκολη[15,29].

2.6 Κρυπτογραφικός αλγόριθμος (μυστικού / δημόσιου κλειδιού)

Για να ξεπεράσουν τα μειονεκτήματα των συμμετρικών αλγορίθμων (ενότητα 1.1.4), οι Diffie, Hellman και Merkle είχαν μία επαναστατική πρόταση που είχε σαν βάση την ακόλουθη ιδέα: Το κλειδί που κατέχει το άτομο που κρυπτογραφεί το μήνυμα (στο παράδειγμα μας η Μαρία), δεν είναι απαραίτητο να είναι μυστικό. Το κρίσιμο σημείο είναι ότι ο Πέτρος, ο παραλήπτης, μπορεί να αποκρυπτογραφεί χρησιμοποιώντας ένα μυστικό κλειδί. Για να καταλάβουμε τη λειτουργία ενός τέτοιου συστήματος, ο Πέτρος εκδίδει ένα δημόσιο κλειδί το οποίο γνωστοποιείτε στον καθένα. Ο Πέτρος έχει επίσης ένα αντίστοιχο μυστικό κλειδί, το οποίο χρησιμοποιείτε για αποκρυπτογράφηση. Έτσι το κλειδί *k* του Πέτρου αποτελείται από δύο μέρη, το δημόσιο k_{pub} και το ιδιωτικό k_{pr}.

Αυτό το σύστημα λειτουργεί όμοια με το παλιό καλό ταχυδρομικό κουτί που συναντάμε στους δρόμους. Ο καθένας μπορεί να τοποθετήσει ένα γράμμα στο κουτί, δηλαδή να κρυπτογραφεί, αλλά μόνο ένα άτομο με το ιδιωτικό-μυστικό κλειδί μπορεί να ανακτά γράμματα, δηλαδή να αποκρυπτογραφεί. Αν υποθέσουμε ότι έχουμε κρυπτοσυστήματα με τέτοια λειτουργικότητα, ένα βασικό πρωτόκολλο για κρυπτογράφηση δημοσίου κλειδιού φαίνεται στην Εικόνα 2.



Εικόνα 8. Βασικό πρωτόκολλο για κρυπτογράφηση δημοσίου κλειδιού[30].

Παρατηρώντας αυτό το πρωτόκολλο ίσως υποστηρίξετε ότι ακόμη και αν μπορούμε να κρυπτογραφήσουμε ένα μήνυμα, χωρίς κάποιο μυστικό κανάλι για την εγκαθίδρυση κλειδιού, παραμένει ακόμα η αδυναμία ανταλλαγής ενός κλειδιού, στη περίπτωση που θα θέλαμε να κρυπτογραφήσουμε, για παράδειγμα, με τον συμμετρικό αλγόριθμο AES. Ωστόσο το πρωτόκολλο μπορεί εύκολα να τροποποιηθεί για αυτή τη χρήση. Αυτό που πρέπει να κάνουμε είναι να κρυπτογραφήσουμε ένα συμμετρικό κλειδί, π.χ., ένα κλειδί AES χρησιμοποιώντας τον αλγόριθμο δημοσίου κλειδιού. Αφού το συμμετρικό κλειδί κρυπτογραφηθεί από τον Πέτρο, και οι δύο πλευρές μπορούν να το χρησιμοποιούν για να κρυπτογραφούν και να αποκρυπτογραφούν μηνύματα χρησιμοποιώντας συμμετρικούς κρυπταλγόριθμους. Η Εικόνα 2. δείχνει ένα βασικό πρωτόκολλο μεταφοράς κλειδιού όπου χρησιμοποιούμε τον συμμετρικό κρυπταλγόριθμο AES για επεξηγηματικούς λόγους (φυσικά, κάποιος μπορεί να χρησιμοποιήσει οποιονδήποτε συμμετρικό αλγόριθμο σε ένα τέτοιο πρωτόκολλο). Το κύριο πλεονέκτημα του πρωτοκόλλου της Εικόνας 1. επί του πρωτοκόλλου της Εικόνας 2. είναι ότι το ωφέλιμο φορτίο είναι κρυπτογραφημένο με ένα συμμετρικό κρυπταλγόριθμο, το οποίο τείνει να είναι πιο γρήγορο από ένα ασύμμετρο αλγόριθμο.

Από τη συζήτηση μέχρι τώρα προκύπτει ότι η ασύμμετρη κρυπτογραφία είναι ένα επιθυμητό εργαλείο για ασφαλής εφαρμογές. Το ερώτημα που παραμένει είναι το πώς μπορεί να δημιουργήσει κάποιος αλγόριθμους δημοσίου κλειδιού; Η απάντηση είναι ότι μπορούν να κατασκευαστούν βάση μίας κοινής αρχής, αυτής της μονόδρομης συνάρτησης (one-way function).

Ορισμός : Μία συνάρτηση *f* είναι μονόδρομη εάν

1. η $y = f(x)$ είναι εύκολη να υπολογιστεί, και
2. η $x = f^{-1}(y)$ είναι ανέφικτη να υπολογιστεί.

Προφανώς, τα επίθετα «εύκολη» και «ανέφικτη» δεν είναι ιδιαίτερα ακριβή. Σε όρους μαθηματικών, μία συνάρτηση είναι εύκολη να υπολογιστεί αν μπορεί να αποτιμηθεί σε πολυωνυμικό χρόνο, δηλαδή ο χρόνος εκτέλεσης της να είναι μία πολυωνυμική έκφραση. Για να είναι χρήσιμη σε πρακτικά συστήματα κρυπτογραφίας, ο υπολογισμός της $y = f(x)$ θα πρέπει να

είναι αρκετά γρήγορος, έτσι ώστε να μην οδηγεί την εφαρμογή σε μη αποδεκτό χρόνο εκτέλεσης. Ο αντίστροφος υπολογισμός $x = f^{1}(y)$ θα πρέπει να είναι τέτοιας υπολογιστικής ισχύος που δε θα είναι εφικτή η αποτίμηση του σε κάθε εύλογο υπολογιστικό διάστημα χρησιμοποιώντας τον βέλτιστο γνωστό αλγόριθμο [18].

Υπάρχουν δύο δημοφιλής one-way συναρτήσεις οι οποίες χρησιμοποιούνται σε πραγματικά συστήματα δημοσίου κλειδιού. Η πρώτη αφορά το πρόβλημα παραγοντοποίησης ακεραίων αριθμών, στο οποίο βασίζεται και ο RSA. Δοθέντος δύο πολύ μεγάλων πρώτων αριθμών, είναι πολύ εύκολο να υπολογίσουμε το γινόμενο τους, ωστόσο είναι πολύ δύσκολο να παραγοντοποιήσουμε το γινόμενο αυτό. Όντως, αν κάθε ένας από τους πρώτους αριθμούς έχει περισσότερα από 150 δεκαδικά ψηφία, το παραγόμενο γινόμενο δε μπορεί να παραγοντοποιηθεί, ακόμη και αν χιλιάδες προσωπικοί υπολογιστές δούλευαν για πολλά χρόνια. Η δεύτερη, η οποία και χρησιμοποιείται ευρέως, σχετίζεται με το πρόβλημα εύρεσης διακριτών λογαρίθμων.

2.6.1 Μηχανισμοί Ασφάλειας

Όπως είδαμε τα συστήματα δημόσιου κλειδιού μπορούν να χρησιμοποιηθούν για κρυπτογράφηση δεδομένων. Αποδεικνύεται ότι μπορούμε να κάνουμε και πολλά άλλα πράγματα, που πριν δε φανταζόμασταν, με τη κρυπτογραφία δημοσίου κλειδιού. Οι κύριες λειτουργίες που μπορούν να μας παρέχουν παρατίθενται παρακάτω:

Εγκαθίδρυση κλειδιού: Υπάρχουν πρωτόκολλα για την εγκαθίδρυση μυστικών κλειδιών μέσω μη ασφαλούς καναλιού. Παραδείγματα τέτοιων πρωτοκόλλων περιλαμβάνουν τον Diffie – Hellman ανταλλαγής κλειδιού (DHKE) ή τον RSA μεταφοράς κλειδιού.

Μη αποποίηση: Η παροχή της μη αποποίησης και της ακεραιότητας του μηνύματος μπορεί να γίνει πραγματικότητα με τους αλγορίθμους ψηφιακών υπογραφών, π.χ., τον DSA ή τον ECDSA.

Ταυτοποίηση: Μπορούμε να ταυτοποιήσουμε οντότητες χρησιμοποιώντας πρωτόκολλα πρόκλησης-απόκρισης μαζί με ψηφιακές υπογραφές, π.χ., σε εφαρμογές όπως οι έξυπνες κάρτες για τραπεζικές συναλλαγές ή για κινητά τηλέφωνα.

Κρυπτογράφηση: Μπορούμε να κρυπτογραφούμε μηνύματα χρησιμοποιώντας αλγορίθμους όπως ο RSA ή ο Elgamal [9].

Σημειώνουμε ότι η ταυτοποίηση και η κρυπτογράφηση μπορούν επίσης να επιτευχθούν με συμμετρικούς κρυπταλγόριθμους, αλλά συνήθως απαιτούν πολύ περισσότερη προσπάθεια ως προς τη διαχείριση των κλειδιών. Φαίνεται ότι τα συστήματα δημοσίου κλειδιού μπορούν να παρέχουν όλες τις λειτουργίες που απαιτούνται από τα μοντέρνα πρωτόκολλα ασφάλειας. Ακόμα κι αν αυτό είναι αλήθεια, στη πράξη, το μεγαλύτερο μειονέκτημα είναι ότι η κρυπτογράφηση δεδομένων απαιτεί μεγάλη υπολογιστική ισχύ με τους αλγόριθμους δημοσίου κλειδιού. Πολλοί κρυπταλγόριθμοι τμήματος αλλά και ροής, μπορούν να κρυπτογραφούν από εκατό φορές μέχρι και χίλιες φορές γρηγορότερα από τους αλγόριθμους δημοσίου κλειδιού. Έτσι, κάπως ειρωνικά θα λέγαμε, η κρυπτογραφία δημοσίου κλειδιού σπάνια χρησιμοποιείται για τη πραγματική κρυπτογράφηση των δεδομένων. Αφετέρου, οι συμμετρικοί αλγόριθμοι είναι φτωχοί στο να παρέχουν τις λειτουργικότητες της «μη αποποίησης» και της εγκαθίδρυσης κλειδιού. Με σκοπό λοιπόν να χρησιμοποιήσουμε το καλύτερο από αυτούς τους δύο κόσμους, τα περισσότερα πραγματικά πρωτόκολλα είναι υβριδικά πρωτόκολλα τα οποία και ενσωματώνουν και τους δύο τύπους αλγορίθμων (συμμετρικούς και δημοσίου κλειδιού). Τέτοια παραδείγματα αποτελούν το πρωτόκολλο SSL/TLS που συνήθως χρησιμοποιούνται για ασφαλές συνδέσεις ιστού, ή το IPsec το οποίο αποτελεί το τμήμα ασφάλειας του πρωτοκόλλου επικοινωνίας του Internet.

2.6.2 Το Παραμένων Πρόβλημα: Η Αυθεντικότητα των Δημοσίων Κλειδιών

Όπως έχουμε δει μέχρι τώρα, το μεγαλύτερο πλεονέκτημα των ασύμμετρων συστημάτων είναι ότι μπορούμε ελεύθερα να διανέμουμε δημόσια κλειδιά. Ωστόσο στη πράξη τα πράγματα είναι πιο δύσκολα καθώς πρέπει να βεβαιώσουμε την αυθεντικότητα των δημοσίων κλειδιών. Με απλά λόγια, ξέρουμε πραγματικά ότι ένα συγκεκριμένο δημόσιο κλειδί ανήκει σε ένα συγκεκριμένο άτομο; Στη πράξη, αυτό το ζήτημα λύνεται με τα ψηφιακά πιστοποιητικά. Χοντρικά

μιλώντας, τα πιστοποιητικά δεσμεύουν ένα δημόσιο κλειδί σε μία ορισμένη οντότητα. Αυτό αποτελεί ένα κυρίαρχο ζήτημα σε πολλά ασφαλή συστήματα, όπως π.χ. στη περίπτωση των συναλλαγών ηλεκτρονικού εμπορίου στο διαδίκτυο. Ένα ακόμα πρόβλημα, το οποίο βέβαια δεν είναι και βασικό, είναι ότι οι αλγόριθμοι δημοσίου κλειδιού απαιτούν πολύ μεγάλα κλειδιά, με αποτέλεσμα τους αργούς ρυθμούς εκτέλεσης.

2.6.3 Σημαντικοί Αλγόριθμοι Δημοσίου Κλειδιού

Σήμερα η κρυπτογραφία έχει φτάσει σε ένα επίπεδο όπου υπάρχουν πάρα πολλοί συμμετρικοί αλγόριθμοι κρυπτογράφησης. Κατά καιρούς έχουν προταθεί διάφοροι, από τους οποίους αρκετοί βρέθηκαν να είναι μη ασφαλής και μερικοί να είναι κρυπτογραφικά ανθεκτικοί. Η κατάσταση όμως είναι εντελώς διαφορετική για τους μη συμμετρικούς αλγόριθμους, καθώς υπάρχουν μόνο τρεις κυρίαρχες οικογένειες αλγορίθμων δημοσίου κλειδιού οι οποίες έχουν πρακτικό αντίκρισμα. Μπορούν να ταξινομηθούν με βάση το υπολογιστικό πρόβλημα που τις χαρακτηρίζουν.

Συστήματα παραγοντοποίησης ακεραίου: Διάφορα συστήματα δημοσίου κλειδιού βασίζονται στο γεγονός ότι είναι πολύ δύσκολο να παραγοντοποιηθούν τεράστιοι ακέραιοι. Ο πιο επιφανής εκπρόσωπος αυτής της οικογένειας είναι ο RSA.

Συστήματα διακριτού λογαρίθμου: Υπάρχουν διάφοροι αλγόριθμοι που βασίζονται σε αυτό που είναι γνωστό ως πρόβλημα διακριτού λογαρίθμου σε πεπερασμένα πεδία. Τα πιο χαρακτηριστικά παραδείγματα αποτελούν ο DH ανταλλαγής κλειδιού, η κρυπτογράφηση Elgamal και ο DSA (Αλγόριθμος Ψηφιακής Υπογραφής).

Συστήματα ελλειπτικών καμπυλών: Μία γενίκευση του αλγόριθμου διακριτού λογαρίθμου είναι τα συστήματα δημοσίου κλειδιού ελλειπτικών καμπυλών. Τα πιο δημοφιλή παραδείγματα περιλαμβάνουν τον ECDH (DH Ελλειπτικών Καμπυλών) και ο ECDSA (Αλγόριθμος Ψηφιακής Υπογραφής Ελλειπτικών Καμπυλών) [6][14].

Οι πρώτες δύο οικογένειες προτάθηκαν στα μέσα της δεκαετίας του 1970, ενώ εκείνη των ελλειπτικών καμπυλών στα μέσα της δεκαετίας του 1980. Μέχρι τώρα δεν υπάρχουν γνωστές επιθέσεις εναντίον αυτών των συστημάτων, αρκεί οι παράμετροι και ειδικά το μήκος του κλειδιού, να επιλεχθούν προσεκτικά. Είναι σημαντικό να σημειώσουμε ότι κάθε μία από αυτές τις τρεις οικογένειες μπορούν να παρέχουν τους βασικούς μηχανισμούς δημοσίου κλειδιού, όπως της εγκαθίδρυσης κλειδιού, της μη αποποίησης μέσω ψηφιακών υπογραφών και της κρυπτογράφησης δεδομένων. Εκτός από τις παραπάνω τρεις οικογένειες, υπήρχαν στο παρελθόν αρκετές προτάσεις για διάφορα άλλα συστήματα δημοσίου κλειδιού, οι οποίες στερούν κρυπτογραφικής ωριμότητας, δηλαδή, δεν είναι γνωστό κατά πόσο ανθεκτικά είναι ενάντια σε μαθηματικές επιθέσεις. Επίσης συνηθισμένο πρόβλημα είναι ότι έχουν φτωχά χαρακτηριστικά υλοποίησης, όπως μέγεθος κλειδιού σε επίπεδο MB, π.χ. τα κρυπτοσυστήματα McEliece. Ωστόσο υπάρχουν επίσης κάποια άλλα συστήματα, για παράδειγμα, κρυπτοσυστήματα υπέρ - ελλειπτικών καμπυλοειδών, τα οποία είναι τόσο αποδοτικά και ασφαλή, όσο και οι καθιερωμένες τρεις οικογένειες που παρουσιάσαμε πιο πάνω, αλλά απλώς δεν έτυχαν ευρείας υιοθέτησης[30,31].

2.7 Διαχείριση κλειδιών - το πρόβλημα της ανταλλαγής κλειδιού - το πρόβλημα της πιστοποίησης κλειδιού

2.7.1 Διαχείριση Κλειδιών

Η διαχείριση κλειδιών είναι η διαδικασία παραγωγής κατάλληλων κλειδιών για ίδια χρήση και διάθεσής τους στους εταίρους με τρόπο ώστε να είναι δυνατόν να επιβεβαιωθούν και να χρησιμοποιηθούν με τον προτιθέμενο τρόπο, ενώ παράλληλα κανείς δεν μπορεί να τα καταχραστεί. Αυτό σημαίνει ότι τα συμμετρικά κλειδιά, καθώς και τα μυστικά κλειδιά της ασύμμετρης κρυπτογραφίας πρέπει να μεταδίδονται εμπιστευτικά (πρόβλημα ανταλλαγής κλειδιού). Για τα δημόσια κλειδιά της ασύμμετρης κρυπτογραφίας πρέπει να είναι δυνατόν να επαληθευτεί η αυθεντικότητα του κλειδιού (πρόβλημα πιστοποίησης κλειδιού).

Οι ίδιες απαιτήσεις ισχύουν για την επικοινωνία μεταξύ μιας αρχής πιστοποίησης και ενός χρήστη για τη διακρίβωση ενός δημόσιου κλειδιού. Είναι απαραίτητη η ανταλλαγή κλειδιών μεταξύ του χρήστη και της αρχής πιστοποίησης. Υπάρχουν δύο κύριες εναλλακτικές προσεγγίσεις για τη λειτουργία της διακρίβωσης δημόσιων κλειδιών, οι οποίες έχουν διαφορετικά χαρακτηριστικά ασφάλειας:

1. Η αρχή πιστοποίησης δημιουργεί το ζεύγος κλειδιών ασύμμετρης κρυπτογραφίας, πιστοποιεί το δημόσιο κλειδί και δίνει στον χρήστη το μυστικό κλειδί και το πιστοποιητικό που περιλαμβάνει το δημόσιο κλειδί.
2. Ο χρήστης δημιουργεί μόνος του το ζεύγος των κλειδιών και αποστέλλει το δημόσιο κλειδί στην αρχή πιστοποίησης, η οποία το πιστοποιεί και δίνει στον χρήστη το πιστοποιητικό που περιλαμβάνει το δημόσιο κλειδί.

Στην πρώτη περίπτωση υφίσταται η ανάγκη για εμπιστευτική επικοινωνία, στη δεύτερη όχι. Η εμπιστευτική επικοινωνία από την αρχή πιστοποίησης προς τον χρήστη είναι εύκολη, με τη δημιουργία μιας έξυπνης κάρτας που περιέχει τα απαραίτητα στοιχεία και που παραδίδεται αυτοπροσώπως στον χρήστη. Αν χρησιμοποιείται ηλεκτρονική μετάδοση, πρέπει να ανταλλαχθεί μεταξύ χρήστη και αρχής πιστοποίησης ένα συμμετρικό κλειδί κρυπτογράφησης με εξωσυστημικό τρόπο. Στην πρώτη περίπτωση η αρχή πιστοποίησης έχει πλήρη έλεγχο για την ποιότητα των παραγόμενων κλειδιών, στη δεύτερη ο κάθε χρήστης είναι υπεύθυνος για το δικό του ζεύγος κλειδιών. Σε κάθε περίπτωση, είναι απαραίτητη μια εξωσυστημική ανταλλαγή πληροφορίας, διαμέσου της οποίας η αρχή πιστοποίησης θα βεβαιωθεί ότι η πιστοποιούμενη οντότητα είναι πραγματικά αυτή που ισχυρίζεται ότι είναι[30,31,32].

2.7.2 Δημόσιοι κατάλογοι

Η χρήση της κρυπτογραφίας δημόσιου κλειδιού καθιστά αναγκαία τη γνώση των δημόσιων κλειδιών των άλλων και την ύπαρξη εμπιστοσύνης προς αυτά. Οι δημόσιοι κατάλογοι, όπως οι κατάλογοι τύπου X500, μπορούν να υποστηρίξουν αυτόν τον στόχο. Οι χρήστες και οι αρχές πιστοποίησης, όταν ενέχονται, έχουν τις ακόλουθες απαιτήσεις όταν χρησιμοποιούν υπηρεσίες ασφάλειας με βάση την κρυπτογραφία δημόσιου κλειδιού:

- Οι χρήστες θέλουν να δημοσιοποιούν τα δικά τους δημόσια κλειδιά, να μπορούν να προσπελάσουν τα πιστοποιητικά των άλλων και τις λίστες ανάκλησης πιστοποιητικών, προκειμένου να διακριβώνουν ότι κάποιο πιστοποιητικό είναι έγκυρο και δεν έχει ανακληθεί.
- Οι αρχές πιστοποίησης επιθυμούν να δημοσιοποιούν τα δημόσια κλειδιά και τις λίστες ανάκλησης με τρόπο που να τα καθιστά διαθέσιμα στην ενδιαφερόμενη κοινότητα.
- Οι αρχές πιστοποίησης επιθυμούν να ανταλλάσσουν κλειδιά αμοιβαίας πιστοποίησης με άλλες αρχές πιστοποίησης και να δημοσιοποιούν τα δικά τους δημόσια κλειδιά.

Οι δημόσιοι κατάλογοι βοηθούν σε αυτή την κατεύθυνση ως παροχείς πληροφορίας στη βασική υποδομή ασφάλειας, αποθηκεύοντας δηλαδή τη σχετική πληροφορία και επιτρέποντας την ευέλικτη αναζήτησή της. Παράλληλα δε, οι κατάλογοι είναι και *χρήστες μηχανισμών ασφάλειας,* καθώς ενσωματώνουν τεχνικές για την προστασία της πληροφορίας του κατάλογου, των επικοινωνιών με άλλες οντότητες καθώς και των πόρων των υπολογιστικών συστημάτων που τους φιλοξενούν. Για τους λόγους αυτούς, οι προδιαγραφές του δημόσιου κατάλογου τύπου X500 έχουν πλαισιωθεί από το *Πλαίσιο Αυθεντικότητας X509*, το οποίο συμπεριλαμβάνει ισχυρούς μηχανισμούς αυθεντικοποίησης[31,32].

2.7.3 Ψηφιακές υπογραφές

Οι ψηφιακές υπογραφές είναι μία τεχνική που επιτρέπει την εξακρίβωση ότι ένα συγκεκριμένο κείμενο προέρχεται από έναν συγκεκριμένο αποστολέα, υποστηρίζοντας έτσι τη διάσταση της *αυθεντικότητας* και επιλύοντας το πρόβλημα της πιστοποίησης κλειδιού. Προκειμένου να εξακριβωθεί η αυθεντικότητα ενός εγγράφου λαμβάνουν χώρα τα κάτωθι βήματα:

1. Αρχικά στο μήνυμα εφαρμόζεται μία *συνάρτηση κερματισμού*, η οποία υπολογίζει ένα πλήθος από bits βάσει του περιεχομένου του μηνύματος. Τα bits αυτά ονομάζονται σύνοψη (digest) του μηνύματος.
2. Η σύνοψη του μηνύματος κρυπτογραφείται με το ιδιωτικό κλειδί του αποστολέα παράγοντας την *ψηφιακή υπογραφή του μηνύματος*.

3. Η ψηφιακή υπογραφή επισυνάπτεται στο μήνυμα και αποστέλλεται μαζί με αυτό.
4. Ο παραλήπτης του μηνύματος αρχικά διαχωρίζει την ψηφιακή υπογραφή από το καθ' εαυτό μήνυμα και εφαρμόζει στο καθ' εαυτό μήνυμα την ίδια συνάρτηση κερματισμού, παράγοντας και αυτός μία σύνοψη.
5. Ακολούθως αποκρυπτογραφεί την ψηφιακή υπογραφή με το δημόσιο κλειδί (σύνοψη) του υπογράψαντος παράγοντα.
6. Τέλος, οι δύο συνόψεις των βημάτων 5 και 6 συγκρίνονται. Μόνο αν αυτά είναι ίσα το μήνυμα θεωρείται αυθεντικό [17].

Σημειώνεται ότι καθώς η ανωτέρω διαδικασία περιλαμβάνει τόσο τα κλειδιά (δημόσιο και ιδιωτικό) του αποστολέα όσο και το ίδιο το περιεχόμενο του μηνύματος (βάσει του οποίου υπολογίζεται το digest), η τεχνική αυτή διακριβώνει τόσο το γεγονός ότι το μήνυμα εστάλη από τον συγκεκριμένο αποστολέα όσο και ότι το περιεχόμενό του παραμένει αναλλοίωτο από τη στιγμή που υπεγράφη ψηφιακά.

3. Κατηγορίες πρωτοκόλλων ανταλλαγής κλειδιού

Ο στόχος ενός πρωτοκόλλου εδραίωσης κλειδιού, που εκτελείται μεταξύ δύο ή περισσότερων συμβαλλόμενων μερών (χρήστες) είναι η απόκτηση ενός κοινού, μυστικού κλειδιού. Το κλειδί αυτό είναι ένα συμμετρικό κλειδί και υπό προϋποθέσεις μπορεί να χρησιμοποιηθεί για κρυπτογράφηση, αυθεντικοποίηση μηνύματος και αυθεντικοποίηση χρήστη.

Στην πράξη, τα κλειδιά *μακράς διαρκείας*, είτε αυτά είναι συμμετρικά είτε έχουν τη μορφή ζεύγους δημόσιου/ιδιωτικού κλειδιού, δε χρησιμοποιούνται απευθείας για κρυπτογραφικές υπηρεσίες. Αντί αυτού, χρησιμοποιούνται ως κλειδιά για την ανταλλαγή κλειδιών (key encrypting keys), για να διευκολύνουν την εδραίωση κλειδιών περιορισμένης χρονικής διάρκειας. Τυπικά, τα πρωτόκολλα εδραίωσης παράγουν εφήμερα κλειδιά που αποκαλούνται *κλειδιά συνόδου*. Η χρήση τους περιορίζεται σε μικρές χρονικές περιόδους μετά το πέρας των οποίων ένα καινούριο κλειδί συνόδου πρέπει να εδραιωθεί ξανά. Οι λόγοι για την εδραίωση κλειδιών συνόδου είναι[30,31,32]:

- Ο περιορισμός της ποσότητας κρυπτογραφημένου υλικού που μπορεί να χρησιμοποιηθεί για κρυπτανάλυση.
- Ο περιορισμός των συνεπειών από την αποκάλυψη ή τη μη εξουσιοδοτημένη πρόσβαση σε κρυπτογραφικά κλειδιά.
- Η άρση της ανάγκης αποθήκευσης πολλών κρυπτογραφικών κλειδιών για μεγάλο χρονικό διάστημα.
- Η ανεξαρτησία μεταξύ των συνόδων επικοινωνίας και μεταξύ των δικτυακών υπηρεσιών.

Τα πρωτόκολλα εδραίωσης κλειδιού μπορούν να κατηγοριοποιηθούν ως εξής:

- **Διανομής κλειδιού**. Μία έμπιστη οντότητα (Κέντρο Διανομής Κλειδιού - ΚΔΚ) συμμετέχει στο πρωτόκολλο εδραίωσης. Στην πιο απλή μορφή των πρωτοκόλλων αυτής της κατηγορίας, το ΚΔΚ επιλέγει το κλειδί συνόδου και το αποστέλλει με ασφαλή τρόπο στους χρήστες, για περαιτέρω επικοινωνία.
- **Μεταφοράς κλειδιού**. Ένας χρήστης δημιουργεί το κλειδί συνόδου και το αποστέλλει με ασφάλεια στον άλλο χρήστη.
- **Συμφωνίας κλειδιού**. Οι χρήστες συμμετέχουν στη δημιουργία του κλειδιού συνόδου, συνεισφέροντας τυχαιότητα ή/και πληροφορία που σχετίζεται με την ταυτότητα τους για την κατασκευή του.

3.1 Εδραίωση κλειδιού με συμμετρική κρυπτογράφηση

Τα πρωτόκολλα που μελετούνται στην ενότητα αυτή, συχνά θεωρούν ότι οι οντότητες που συμμετέχουν σε ένα σύστημα εδραίωσης μοιράζονται εκ των προτέρων συμμετρικά κλειδιά μακράς διαρκείας. Τα κλειδιά αυτά έχουν προ-διανεμηθεί στους χρήστες του συστήματος, τυπικά από ένα ΚΔΚ. Γενικότερα, η απαίτηση για ύπαρξη προ-εγκατεστημένων κλειδιών μεταξύ των χρηστών του συστήματος αποτελεί μειονέκτημα των πρωτοκόλλων αυτής της κατηγορίας.

Στο Κεφάλαιο αυτό, όπως και στο επόμενο, θα χρησιμοποιηθούν κάποιοι συμβολισμοί για λόγους πρακτικότητας και βέλτιστης απεικόνισης:

- id_A: Μοναδική ταυτότητα του χρήστη *A*
- r_U: Τυχαιότητα που επιλέγεται από τον χρήστη *U*
- N_U: Μοναδικός αριθμός που επιλέγεται από τον χρήστη *U*
- t_U: Χρονοσφραγίδα που επιλέχθηκε από τον χρήστη *U*
- K_{AB}: Συμμετρικό κλειδί που μοιράζονται οι χρήστες *A* και *B*
- K_S: Συμμετρικό κλειδί συνόδου
- $E_K(m)$ Συμμετρική κρυπτογράφηση του *m* με το κλειδί *K*
- SK_A, PK_A Ζεύγος ιδιωτικού – δημόσιου κλειδιού κρυπτογράφησης
- $Cert_X$ Πιστοποιητικό που δένει την οντότητα *X* με ένα δημόσιο κλειδί κρυπτογράφησης ή/και υπογραφής

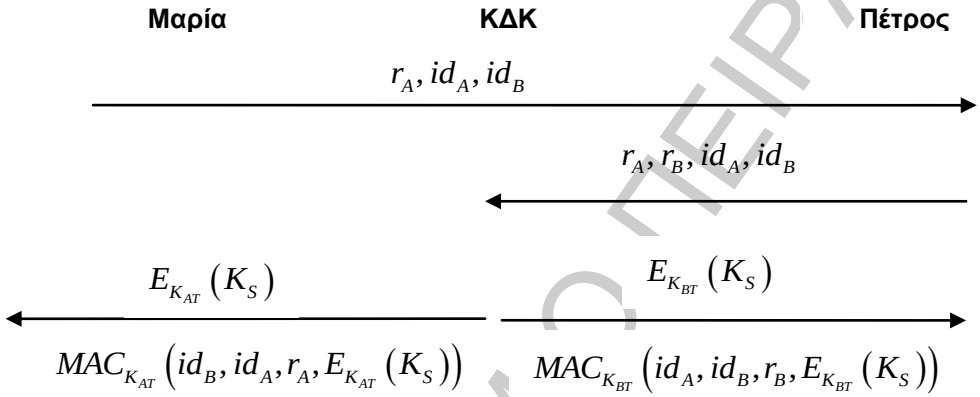
3.1.1 Διανομή κλειδιού με συμμετρικές τεχνικές

Έστω η Μαρία αιτείται από το ΚΔΚ ένα κλειδί συνόδου για επικοινωνία με τον Πέτρο [25]. Το ΚΔΚ επιλέγει ένα κλειδί συνόδου K_S και στέλνει στη Μαρία δύο αντίγραφα του κλειδιού:

- Ένα αντίγραφο $E_{K_{AT}}$ (K_S) κρυπτογραφημένο με το κλειδί K_{AT} που μοιράζεται με τη Μαρία,
- Ένα αντίγραφο $E_{K_{BT}}$ (K_S) κρυπτογραφημένο με το κλειδί K_{BT} που μοιράζεται με τον Πέτρο.

Η Μαρία αποκρυπτογραφεί το αντίγραφο της και στέλνει στον Πέτρο το αντίγραφο του. Ο Πέτρος αποκρυπτογραφεί το αντίγραφο του για να αποκτήσει το κλειδί K_S και στη συνέχεια η Μαρία και Ο Πέτρος επικοινωνούν με το κλειδί K_S.

Το 1995 οι Bellare και Rogaway πρότειναν το πρωτόκολλο διανομής που απεικονίζεται στην Εικόνα 9.



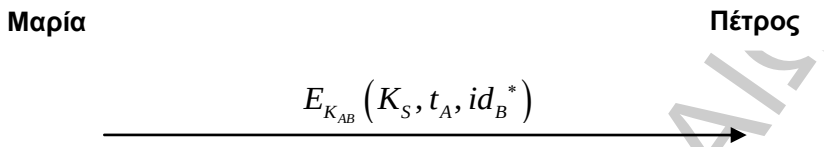
Εικόνα 9. Το πρωτόκολλο Bellare και Rogaway[26].

1. Η Μαρία επιλέγει έναν τυχαίο αριθμό r_A και το στέλνει στον Πέτρο μαζί με τις ταυτότητες id_A, id_B.
2. Ο Πέτρος επιλέγει έναν τυχαίο αριθμό r_B και το προωθεί, μαζί με το μήνυμα της Μαρίας, στο ΚΔΚ.
3. Το ΚΔΚ επιλέγει το κλειδί συνόδου K_S, και το κρυπτογραφεί με τα κλειδιά που μοιράζεται με τον Πέτρο και τη Μαρία, K_{BT} και K_{AT} αντίστοιχα. Στη συνέχεια φτιάχνει ένα μήνυμα που αποτελείται από τις ταυτότητες id_A και id_B, την τυχαιότητα r_B και το κρυπτογράφημα $E_{K_{BT}}(K_S)$, υπολογίζει την τιμή MAC του μηνύματος και τη στέλνει μαζί με το $E_{K_{BT}}(K_S)$ στον Πέτρο. Ομοίως υπολογίζει τα $E_{K_{AT}}(K_S)$, την αντίστοιχη τιμή MAC και τα στέλνει στη Μαρία.

Το πρωτόκολλο ολοκληρώνεται με επιτυχία όταν η Μαρία και ο Πέτρος επαληθεύουν τις τιμές MAC που λαμβάνουν στο βήμα 3 και προσφέρει εννοούμενη αυθεντικοποίηση κλειδιού. Από τη μεριά της Μαρίας για παράδειγμα, η τιμή MAC περιέχει τη τιμή r_A που η ίδια έστειλε στο βήμα 1., καθώς επίσης τις ταυτότητες της Μαρίας και του Πέτρου και την τιμή $E_{K_{AT}}(K_S)$. Η τιμή MAC πιστοποιεί πως το μήνυμα έχει δημιουργηθεί από το ΚΔΚ, εφόσον το ΚΔΚ είναι ο μόνος (εκτός της Μαρίας) που γνωρίζει το συμμετρικό κλειδί για τον υπολογισμό της. Το γεγονός πως η τιμή MAC αποτρέπει έναν παρείσακτο από το να αντικαταστήσει το κλειδί συνόδου που δημιούργησε το ΚΔΚ με κάποιο άλλο. Επειδή μόνο η Μαρία μπορεί να αποκρυπτογραφήσει το $E_{K_{AT}}(K_S)$ κι εφόσον η ταυτότητα του Πέτρου βρίσκεται επίσης στην τιμή MAC, η Μαρία πείθεται ότι ο Πέτρος είναι ο μόνος ικανός να αποκτήσει πρόσβαση στο κλειδί συνόδου K_S. Τα ίδια επιχειρήματα ισχύουν από τη σκοπιά του Πέτρου.

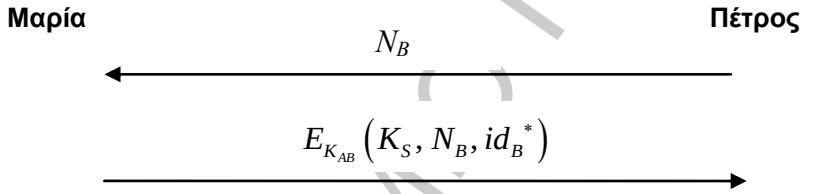
3.1.2 Μεταφορά κλειδιού με συμμετρική κρυπτογράφηση

Έστω η Μαρία και ο Πέτρος μοιράζονται ένα συμμετρικό κλειδί μακράς διαρκείας K_{AB}. Ένα πρωτόκολλο απλής μεταφοράς κλειδιού φαίνεται στην Εικόνα 10. [26] Το κλειδί συνόδου είναι η τιμή K_S. Η χρονοσφραγίδα t_A στο μήνυμα που αποστέλλει η Μαρία, εισάγεται για την αποφυγή επιθέσεων επανάληψης. Η ταυτότητα του Πέτρου, id_B, εισάγεται για να αποτραπούν επιθέσεις αντανάκλασης, δηλαδή απευθείας επανάληψης του μηνύματος στη Μαρία. Το πρωτόκολλο παρέχει μονόδρομη αυθεντικοποίηση χρήστη (της Μαρίας από τον Πέτρο) και έμμεση αυθεντικοποίηση κλειδιού.



Εικόνα 10. Πρωτόκολλο απλής μεταφοράς κλειδιού[26].

Ένα ακόμα πρωτόκολλο αυτής της κατηγορίας είναι η μεταφορά κλειδιού με πρόσκληση – απάντηση. Με τη χρήση μοναδικών αριθμών και κόστος ένα επιπλέον βήμα, εξαλείφεται η ανάγκη συγχρονισμού λόγω χρονοσφραγίδων (Εικόνα 11.).

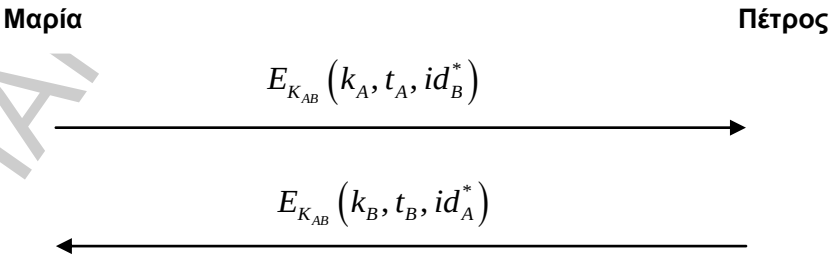


Εικόνα 11. Απλή μεταφορά κλειδιού με πρόσκληση – απάντηση[26].

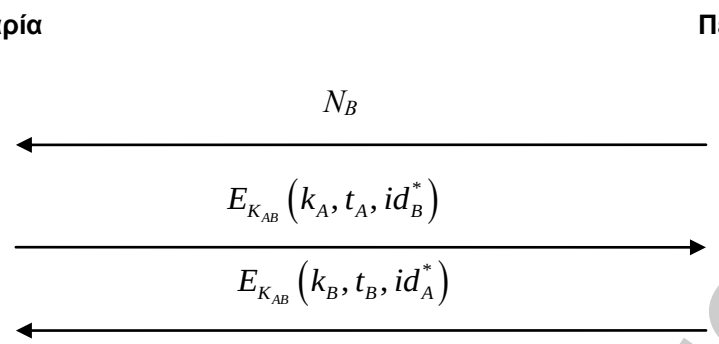
3.1.3 Συμφωνία κλειδιού με συμμετρική κρυπτογράφηση

Τα δύο προηγούμενα πρωτόκολλα που παρουσιάσαμε (απλή μεταφορά, μεταφορά με πρόσκληση - απάντηση) μπορούν εύκολα να τροποποιηθούν ώστε να πετύχουν συμφωνία κλειδιού κάνοντας χρήση χρονοσφραγίδων. Για παράδειγμα στο πρωτόκολλο της Εικόνας 12. η Μαρία με τον Πέτρο συνεισφέρουν και ανταλλάσουν τις τυχαιότητες k_A, k_B. Οι τιμές αυτές μπορούν να δοθούν ως είσοδος σε μια συνάρτηση κατακερματισμού H, ώστε να προκύψει το κλειδί συνόδου $K_S = H(k_A, k_B)$.

Κάνοντας χρήση μοναδικών αριθμών στα πλαίσια μιας τεχνικής πρόσκλησης-απάντησης, το πρωτόκολλο της Εικόνας 12., μπορεί να τροποποιηθεί σύμφωνα με την απεικόνιση της Εικόνας 13.



Εικόνα 12. Συμφωνία κλειδιού με χρονοσφραγίδες[26].



Εικόνα 13. Συμφωνία κλειδιού με πρόσκληση – απάντηση[26].

3.2 Εδραίωση κλειδιού με κρυπτογραφία δημόσιου κλειδιού

Στα πρωτόκολλα αυτής της ενότητας δεν απαιτείται η ύπαρξη προ-εγκατεστημένων συμμετρικών κλειδιών μεταξύ των χρηστών του συστήματος, επιλύοντας έτσι ένα από τα σημαντικότερα προβλήματα διαχείρισης κλειδιού σε συμμετρικά συστήματα. Ωστόσο απαιτείται η ύπαρξη μιας υποδομής δημόσιου κλειδιού που πιστοποιεί τα έγκυρα δημόσια κλειδιά των χρηστών του συστήματος.

3.2.1 Μεταφορά κλειδιού με τεχνικές δημοσίου κλειδιού

Το 1974, ο Merkle διατύπωσε ένα σύστημα γνωστό και ως *Γρίφοι του Merkle* [27] για την εδραίωση κλειδιού μεταξύ δύο χρηστών, χωρίς την προϋπόθεση οι χρήστες να μοιράζονται εκ των προτέρων μυστική πληροφορία. Αφ' υψηλού, η Μαρία ετοιμάζει ℓ γρίφους, κάθε ένας από τους οποίους απαιτεί *O*(*n*) υπολογιστικά βήματα για την επίλυση του. Η Μαρία στέλνει τους γρίφους στον Πέτρο, μέσα από ένα μη προστατευμένο (ανοικτό) κανάλι επικοινωνίας. Ο Πέτρος επιλέγει έναν από τους γρίφους, τον επιλύει με κόστος *O*(*n*) υπολογισμούς, και στη συνέχεια χρησιμοποιεί το αποτέλεσμα ως το κλειδί συνόδου για να κρυπτογραφήσει και να στείλει στην Μαρία ένα προκαθορισμένο μήνυμα. Το μήνυμα αυτό η Μαρία θα το αποκρυπτογραφήσει με κόστος *O(1)* υπολογισμούς.

Η ιδέα του Merkle είναι ενδιαφέρουσα: Ενώ η Μαρία και ο Πέτρος εδραιώνουν το κλειδί με πολυπλοκότητα *O*(*n*), ένας παθητικός εχθρός (παρείσακτος) θα χρειαστεί $O(\ln)$ υπολογισμούς για να σπάσει το σύστημα. Επομένως, το σύστημα θεωρείται ασφαλές όταν οι τιμές ℓ, *n* επιλέγονται ώστε το *O*(*n*) να είναι υπολογιστικά εφικτό, ενώ το $O(\ln)$ να μην είναι, θεωρώντας πολυωνυμικούς εχθρούς.

Έστω λοιπόν η Μαρία διαθέτει μια βάση δεδομένων με $\ell = 2^{20}$ μηνύματα της μορφής $m_i = \left(K_i, SN_i\right), i = 1, 2, ..., 2^{20}$, όπου K_i είναι ένα κλειδί συνόδου και SN_i είναι ένας σειριακός αριθμός. Το πρωτόκολλο εδραίωσης λειτουργεί ως εξής:

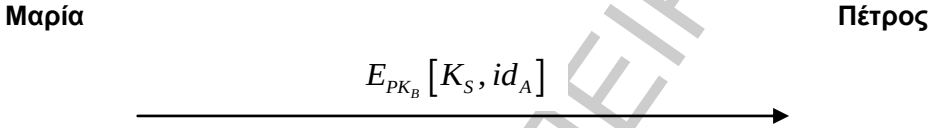
1. Η Μαρία κρυπτογραφεί κάθε μήνυμα $m_i, i = 1, 2, \ldots, 2^{20}$, με ένα κλειδί k_i σχετικά μικρού μήκους π.χ. 30 bit. Η Μαρία στέλνει στον Πέτρο τα 2^{20} κρυπτογραφημένα μηνύματα: $E_{k_i}\left(m_i\right), i = 1, 2, \ldots, 2^{20}$.
2. Ο Πέτρος επιλέγει στην τύχη ένα μήνυμα και εκτελεί μια εξαντλητική αναζήτηση, για να βρει το κλειδί, με κόστος $\frac{1}{2}2^{30} = 2^{29}$ υπολογισμούς κατά μέσο όρο. Έστω ο Πέτρος ανακτά το μήνυμα $m_j = \left(K_j, SN_j\right)$.
3. Ο Πέτρος κρυπτογραφεί ένα προκαθορισμένο μήνυμα *m* με το κλειδί K_j και στέλνει στη Μαρία $E_{K_j}\left(m\right), SN_j$. Το εδραιωμένο κλειδί είναι το K_j.

Το πρωτόκολλο δεν είναι ιδιαίτερα αποδοτικό, ωστόσο αποτελεί ίσως το πρώτο παράδειγμα εδραίωσης κλειδιού με ασύμμετρες τεχνικές. Η ασυμμετρία του πρωτοκόλλου συνίσταται στο εξής: Ο παθητικός εχθρός δεν ξέρει πιο μήνυμα m_j αντιστοιχεί στο σειριακό αριθμό SN_j. Επομένως θα πρέπει να εκτελέσει μία εξαντλητική αναζήτηση σε κάθε ένα από τα 2^{20} κρυπτογραφήματα $E_{k_i}(m_i)$, με συνολικό κόστος: $(2^{19} \times 2^{29}) = 2^{48}$.

Έστω τώρα ένα απλό πρωτόκολλο μεταφοράς το οποίο συνοψίζεται στα εξής βήματα:

1. Η Μαρία αποκτά, με κάποιο τρόπο, το έγκυρο δημόσιο κλειδί του Πέτρου, PK_B.
2. Η Μαρία επιλέγει ένα κλειδί συνόδου K_S, το κρυπτογραφεί μαζί με την ταυτότητα της id_A, με το κλειδί PK_B και το στέλνει στον Πέτρο.
3. Ο Πέτρος χρησιμοποιεί το ιδιωτικό του κλειδί, SK_B, αποκρυπτογραφεί το μήνυμα που έλαβε στο βήμα 2 και αποκτά το κλειδί K_S.

Η Μαρία και Ο Πέτρος χρησιμοποιούν το κλειδί K_S για την επικοινωνία τους, κάνοντας χρήση ενός συμμετρικού αλγορίθμου κρυπτογράφησης. Το πρωτόκολλο ουσιαστικά υλοποιείται σε 1 βήμα, όπως φαίνεται στην Εικόνα 14.

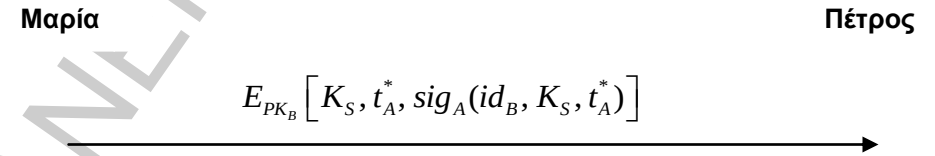


Εικόνα 14. Ένα πρωτόκολλο μεταφοράς κλειδιού[28].

Στα πρωτόκολλα εδραίωσης συχνά υποθέτουμε ότι οι χρήστες είναι γνώστες των αυθεντικών δημόσιων κλειδιών των χρηστών με τους οποίους συνομιλούν. Στο παράδειγμα μας, το κλειδί PK_B μπορεί να είναι δημοσιευμένο σε μια βάση δεδομένων, ή να πιστοποιείται από μια Αρχή Πιστοποίησης, στα πλαίσια μιας ΥΔΚ στην οποία ανήκουν οι χρήστες του συστήματος. Σε αυτήν την περίπτωση το πρωτόκολλο αποκτά ένα επιπλέον βήμα: αρχικά ο Πέτρος στέλνει στην Μαρία ένα πιστοποιητικό, υπογεγραμμένο από μια αρχή πιστοποίησης που εμπιστεύονται και οι δύο, στο οποίο αναγράφεται το PK_B και η ταυτότητα του Πέτρου, id_B.

Το παραπάνω πρωτόκολλο παρέχει εννοούμενη αυθεντικοποίηση κλειδιού για την Μαρία (μόνον ο Πέτρος, που έχει το αντίστοιχο ιδιωτικό κλειδί μπορεί να αποκτήσει πρόσβαση στο *KS*). Ωστόσο, στο παραπάνω πρωτόκολλο, η Μαρία δεν μπορεί να αυθεντικοποιήσει την ταυτότητα του Πέτρου. Ο Πέτρος επίσης δεν επιβεβαιώνει την πηγή του μηνύματος που λαμβάνει, ούτε αν η Μαρία είναι ενεργή στο πρωτόκολλο.

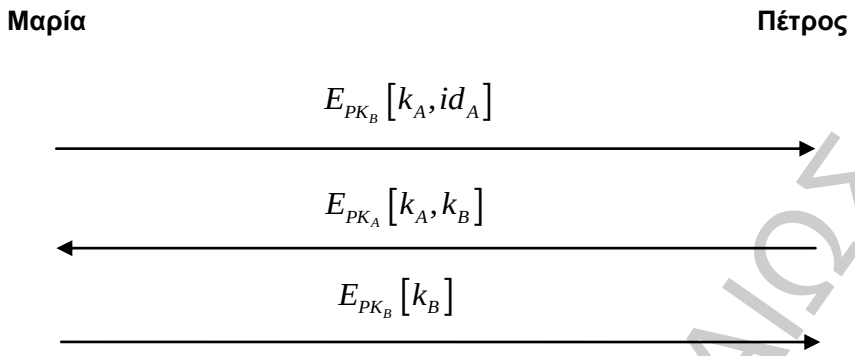
Στην υβριδική αυτή προσέγγιση, ελαφρώς διαφοροποιημένη, βασίζονται συστήματα όπως το πρότυπο TLS/SSL. Το πρωτόκολλο μπορεί να ενισχυθεί με τεχνικές αυθεντικοποίησης χρήστη, ώστε ο Πέτρος να ξέρει ότι συνομιλεί με την Μαρία (και αντιστρόφως). Για παράδειγμα, το πρωτόκολλο στην Εικόνα 15. κάνει χρήση ψηφιακών υπογραφών και χρονοσφραγίδων [28].



Εικόνα 15. Μεταφορά κλειδιού με αυθεντικοποίηση οντότητας[28].

3.2.2 Συμφωνία κλειδιού με τεχνικές δημόσιου κλειδιού

Οι Needham και Schroeder προτείνουν [29] ένα απλό πρωτόκολλο συμφωνίας κλειδιού μεταξύ της Μαρίας και του Πέτρου, που απεικονίζεται στην Εικόνα 16., για την επίτευξη αμοιβαίας αυθεντικοποίησης οντότητας.

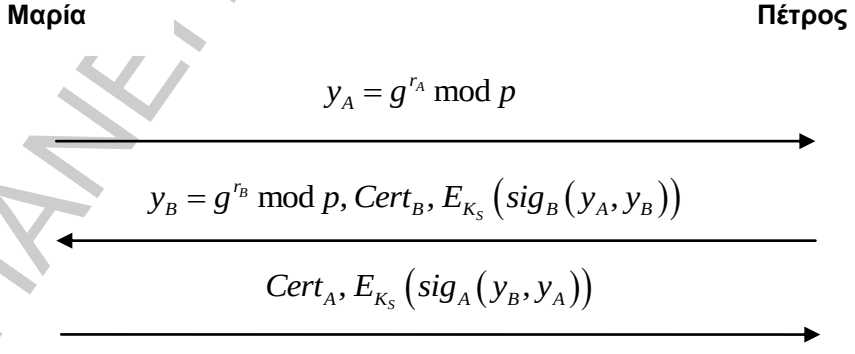


Εικόνα 16. Συμφωνία κλειδιού με αμοιβαία αυθεντικοποίηση[29].

1. Η Μαρία κρυπτογραφεί μια τυχαία τιμή k_A καθώς και την ταυτότητα της id_A, με το δημόσιο κλειδί του Πέτρου, PK_B και στέλνει το κρυπτογράφημα στον Πέτρο.
2. Ο Πέτρος αποκρυπτογραφεί το μήνυμα με το ιδιωτικό του κλειδί, SK_B. Στη συνέχεια επιλέγει μια τυχαία τιμή k_B και την κρυπτογραφεί, μαζί με το k_A, με το δημόσιο κλειδί της Μαρίας, PK_A και στέλνει το κρυπτογράφημα στη Μαρία.
3. Η Μαρία αποκρυπτογραφεί το κρυπτογράφημα με το ιδιωτικό της κλειδί SK_A και ελέγχει αν το k_A ταυτίζεται με την τιμή που έστειλε στο βήμα 1. Αν ναι, κρυπτογραφεί το k_B με το PK_B και στέλνει το κρυπτογράφημα στο Πέτρο. Ο Πέτρος αποκρυπτογραφεί και ελέγχει αν το k_B υπάρχει στο μήνυμα.

Το κλειδί συνόδου είναι μια συνάρτηση των τιμών k_A και k_B, για παράδειγμα: $K_S = H(k_A, k_B)$, που και οι δύο χρήστες μπορούν να υπολογίσουν. Το πρωτόκολλο παρέχει αμοιβαία αυθεντικοποίηση οντότητας και κλειδιού. Τονίζεται πως οι τυχαιότητες k_A και k_B, εκτός από την αυθεντικοποίηση οντότητας, χρησιμοποιούνται και για τη δημιουργία του κλειδιού συνόδου.

Το πρωτόκολλο συμφωνίας κλειδιού Diffie – Hellman που μελετάμε στην ενότητα 3.4.1 ανήκει σε αυτή τη κατηγορία πρωτοκόλλων. Μια επέκταση του που προτάθηκε από τους Diffie, Van Oorschot και Wiener αποτελεί ο αλγόριθμος *Station to Station (STS)*. Σε αυτόν τον αλγόριθμο, το πρόβλημα της ενδιάμεσης οντότητας στα πρωτόκολλα ανταλλαγής κλειδιού μπορεί να επιλυθεί χρησιμοποιώντας τεχνικές αυθεντικοποίησης. Γίνεται χρήση ψηφιακών υπογραφών και πιστοποιητικών δημόσιου κλειδιού. Μια απλοποιημένη περιγραφή λειτουργίας του παρουσιάζεται στην Εικόνα 17. Το πρωτόκολλο IKE (Βλέπε 3.7.5) που χρησιμοποίει η σουίτα IPSec (Βλέπε 3.7.2) για ανταλλαγή κλειδιών, αποτελεί μια παραλλαγή του πρωτοκόλλου STS.



Εικόνα 17. Το πρωτόκολλο STS[28].

Υποθέτουμε εδώ ότι η Μαρία και ο Πέτρος έχουν στην κατοχή τους τα έγκυρα πιστοποιητικά του δημόσιου κλειδιού τους, $Cert_A$ και $Cert_B$ αντίστοιχα, υπογεγραμμένα από μια αρχή πιστοποίησης που και οι δύο εμπιστεύονται.

1. Η Μαρία επιλέγει έναν τυχαίο εκθέτη $r_A \in Z_p^*$ και στέλνει στο Πέτρο την τιμή $y_A = g^{r_A} \bmod p.$
2. Ο Πέτρος επιλέγει έναν τυχαίο εκθέτη $r_B \in Z_p^*,$ υπολογίζει το κλειδί συνόδου $K_S = \left(g^{r_A}\right)^{r_B} \bmod p$ και στέλνει στη Μαρία την τιμή $y_B = g^{r_B} \bmod p,$ το πιστοποιητικό $Cert_B$, καθώς και ένα μήνυμα κρυπτογραφημένο με το κλειδί K_S, που περιέχει τις τιμές y_A, y_B ψηφιακά υπογεγραμμένες με το ιδιωτικό κλειδί του Πέτρου.
3. Η Μαρία ομοίως στέλνει στον Πέτρο το πιστοποιητικό $Cert_A$, καθώς κι ένα μήνυμα κρυπτογραφημένο με το κλειδί K_S, που περιέχει τις τιμές y_A, y_B, ψηφιακά υπογεγραμμένες με το ιδιωτικό της κλειδί.

Το κλειδί συνόδου είναι το: $K_S \equiv g^{r_A r_B} \equiv \left(g^{r_A}\right)^{r_B} \equiv \left(g^{r_B}\right)^{r_A} (\bmod p).$

Το πρωτόκολλο STS προσφέρει αμοιβαία αυθεντικοποίηση χρήστη και κλειδιού. Το πρωτόκολλο προσφέρει επίσης ανωνυμία και αμοιβαία επιβεβαίωση κλειδιού, καθώς και πρόσθια μυστικότητα. Στο πρωτόκολλο οι τιμές y_A και y_B λειτουργούν εκτός των άλλων, και ως προκλήσεις για την αμοιβαία αυθεντικοποίηση του Πέτρου και της Μαρίας. Στο βήμα 3 η Μαρία επαληθεύει την υπογραφή του Πέτρου στο μήνυμα και ότι το μήνυμα περιέχει την τιμή y_A που είχε στείλει στο βήμα 1. Ομοίως πράττει και ο Πέτρος μετά την ολοκλήρωση του βήματος 3, ώστε να αυθεντικοποιήσει την Μαρία. Το γεγονός αποτρέπει το ενεργητικό εχθρό (παρείσακτο) από επιθέσεις πλαστοπροσωπίας ή/και ενδιάμεσης οντότητας. Ως προς τον παθητικό εχθρό, η Μαρία και ο Πέτρος συμφωνούν στο κλειδί συνόδου όπως και στο πρωτόκολλο Diffie - Hellman, όπου η μυστικότητα του κλειδιού ανάγεται στη δυσκολία επίλυσης του προβλήματος DDH, στο σημασιολογικό μοντέλο.

3.3 Ανταλλαγή κλειδιού με RSA

Το 1977 οι Ron Rivest, Adi Shamir και Len Adleman πρότειναν έναν αλγόριθμο, γνωστό ως RSA, ο οποίος είναι ένας από τους πιο διαδεδομένους και περισσότερο χρησιμοποιημένους αλγόριθμους στην κρυπτογραφία δημοσίου κλειδιού. Αυτός ο αλγόριθμος είναι κατάλληλος για κρυπτογράφηση/αποκρυπτογράφηση δεδομένων, για την δημιουργία ψηφιακών υπογραφών και την επαλήθευσή τους καθώς και για την ασφαλή μεταφορά κλειδιών. Χρησιμοποιείται ως βάση για τη δημιουργία μιας ασφαλούς γεννήτριας ψευδοτυχαίων αριθμών καθώς και για την ασφάλεια σε ορισμένα ηλεκτρονικά παιχνίδια. Ο RSA βασίζεται στις αρχές της θεωρίας αριθμών (Ενότητα 2.1). Στη συνέχεια αναφέρονται τα βήματα που ακολουθούνται για την υλοποίηση του αλγορίθμου:

1. Επιλέγονται δύο μεγάλοι πρώτοι αριθμοί, p και q (συνήθως πολύ μεγαλύτεροι από 10^{100})
2. Υπολογίζεται $n = p \cdot q$ και $\phi(n) = (p-1) \cdot (q-1)$. Ο n ονομάζεται *υπόλοιπο RSA (RSA modulus)*
3. Επιλέγεται ο δημόσιος εκθέτης $e \in \{1, 2, \ldots, \phi(n)-1\}$ τέτοιος ώστε $\gcd(e, \Phi(n)) = 1$
4. Υπολογίζεται το μυστικό κλειδί d τέτοιο ώστε $d \cdot e \equiv \bmod \phi(n)$.
5. Το δημόσιο κλειδί αποτελείται από το ζευγάρι $k_{pub} = (n, e)$ και το ιδιωτικό $k_{pr} = (d)$.

Έχοντας υπολογίσει εκ των προτέρων τις παραπάνω παραμέτρους ξεκινά η κρυπτογράφηση. Τα βήματα που ακολουθούνται για την κρυπτογράφηση ενός κειμένου m περιγράφονται παρακάτω:

1. Το κείμενο το οποίο θα κρυπτογραφηθεί (που θεωρείται ως ακολουθία bit) διαιρείται σε μπλοκ, έτσι ώστε κάθε μήνυμα κειμένου, m, να πέφτει στο διάστημα $0 \le m < n$. Αυτό μπορεί να γίνει με ομαδοποίηση του κειμένου σε μπλοκ των k bit, όπου το k είναι ο μεγαλύτερος ακέραιος για τον οποίο η σχέση $2^k < n$ είναι αληθής.
2. Υπολογίζεται το $c = m^e (\bmod n)$ όπου το c είναι το κρυπτογραφημένο κείμενο.

Για την αποκρυπτογράφηση του μηνύματος c ακολουθούνται τα παρακάτω βήματα:

1. Υπολογίζεται το $m = c^d (\mathrm{mod} n)$ όπου το *m* είναι το αρχικό κείμενο.

Η ασφάλεια της μεθόδου οφείλεται στην δυσκολία της παραγοντοποίησης μεγάλων αριθμών. Εάν ο κρυπτοαναλυτής μπορούσε να παραγοντοποιήσει το δημόσια γνωστό *n*, θα μπορούσε στη συνέχεια να βρει τα *p* και *q* και από αυτά το $\phi(n)$. Αν διαθέτει τα $\phi(n)$ και *e* μπορεί να βρει το *d* με τη βοήθεια του αλγορίθμου του Ευκλείδη. Σύμφωνα με τον Rivest και τους συναδέλφους του, η παραγοντοποίηση ενός αριθμού με 200 ψηφία απαιτεί 4 δισεκατομμύρια χρόνια υπολογιστικού χρόνου θεωρώντας ότι γίνεται χρήση ενός υπολογιστή με χρόνο εντολής 1 μsec. Ακόμα και αν οι υπολογιστές συνεχίσουν να γίνονται ταχύτεροι κατά μία τάξη μεγέθους ανά δεκαετία θα χρειαστούν αιώνες για να γίνει δυνατή η παραγοντοποίηση αριθμών με 500 ψηφία, αλλά και τότε θα μπορούμε απλά να επιλέγουμε μεγαλύτερα *p* και *q* . Το σημερινό επίπεδο της έρευνας πάνω στην παραγοντοποίηση των αριθμών απαιτεί τα κλειδιά που παράγονται με τον αλγόριθμο RSA να έχουν μήκος τουλάχιστον 1024 bit έτσι ώστε να παρέχεται ικανοποιητική ασφάλεια στις επικοινωνίες μέσα στα επόμενα χρόνια.

3.3.1 Πλεονεκτήματα του RSA

Ο RSA, σαν αλγόριθμος ανταλλαγής και εγκαθίδρυσης κρυπτογραφικών κλειδιών, παρέχει μερικά πλεονεκτήματα τα οποία βοήθησαν στην υλοποίηση πιο ασφαλών και ευκολότερα διαχειρίσιμων συναλλαγών. Τα πλεονεκτήματα αυτά περιλαμβάνουν:

- *Απλοποίηση του προβλήματος της διαχείρισης κλειδιών*: Στην συμμετρική κρυπτογραφία ο αριθμός των κλειδιών που απαιτείται για την επικοινωνία *n* οντοτήτων σε ένα κρυπτοσύστημα είναι ανάλογος του n^2. Στην ασύμμετρη κρυπτογραφία όμως κάθε χρήστης χρειάζεται δύο κλειδιά, έτσι ο απαιτούμενος αριθμός κλειδιών είναι απλά $2n$. Γίνεται κατανοητό λοιπόν ότι σε ένα κρυπτοσύστημα δημοσίου κλειδιού η σχέση που συνδέει τον αριθμό των χρηστών με τον αριθμό των κλειδιών είναι γραμμική και για αυτό το λόγο εύκολα διαχειρίσιμη ακόμα και όταν ο αριθμός των χρηστών είναι πολύ μεγάλος.
- *Ενισχυμένη ασφάλεια των συναλλαγών*: Κάθε χρήστης παράγει μόνος του και για δική του χρήση ένα ζεύγος κλειδιών. Το ιδιωτικό κλειδί θα πρέπει να μένει μυστικό και κρυφό από οποιαδήποτε μη εξουσιοδοτημένη οντότητα εξαλείφοντας έτσι όχι μόνο το πρόβλημα της μεταφοράς του αλλά και την απαίτηση για την εγκατάσταση ενός ασφαλούς διαύλου επικοινωνίας. Το δημόσιο κλειδί από την άλλη είναι ευρέως διαθέσιμο και άρα μπορεί να μεταφερθεί με οποιαδήποτε βολική μέθοδο σε ένα δίκτυο χωρίς να τίθεται θέμα για την διατήρηση της μυστικότητάς του.

Ο αλγόριθμος RSA είναι κάτι παραπάνω από δεδομένο στην κρυπτογραφία δημοσίου κλειδιού σε σημείο μάλιστα που οι δύο έννοιες να θεωρούνται ταυτόσημες. Η ισχύς του RSA είναι τόσο μεγάλη που η κυβέρνηση των ΗΠΑ έχει περιορίσει σημαντικά την εξαγωγή του αλγορίθμου σε ξένες χώρες.

3.3.2 Πιθανές επιθέσεις στον RSA

Αν και ο αλγόριθμος RSA είναι ο επικρατέστερος στο χώρο της κρυπτογραφίας δημοσίου κλειδιού έχει κάποιες αδυναμίες. Μερικά από τα πιο σημαντικά προβλήματα που θα μπορούσε να αντιμετωπίσει ο αλγόριθμος αυτός είναι τα παρακάτω:

- *Παραγοντοποίηση του δημοσίου κλειδιού*: Αυτό που θα ήθελε φυσικά να πετύχει κάθε εισβολέας σε ένα δίκτυο επικοινωνίας είναι να ανακτήσει το αρχικό κείμενο *m* από το αντίστοιχο κρυπτοκείμενο *c*, δοσμένου του δημοσίου κλειδιού (*e,n*) του παραλήπτη. Το παραπάνω είναι γνωστό και ως *πρόβλημα RSA (RSA problem, RSAP)*. Μια πιθανή προσέγγιση που θα μπορούσε να υλοποιήσει ο εισβολέας για να λύσει το πρόβλημα RSA είναι αρχικά να παραγοντοποιήσει το *n* και στη συνέχεια να υπολογίσει τα $\phi(n)$ και *d* ακριβώς όπως έκανε και ο χρήστης του δικτύου. Από τη στιγμή που ο εισβολέας καταφέρει να υπολογίσει το *d* τότε μπορεί να αποκρυπτογραφήσει οποιαδήποτε πληροφορία στέλνεται σε αυτόν το χρήστη. Το παραπάνω σενάριο πρόκειται για την πιο μεγάλη απειλή που μπορεί να αντιμετωπίσει ο αλγόριθμος αυτός. Προς το παρόν ο RSA φαίνεται να είναι εξαιρετικά δυνατός και ασφαλής. Αν και, όπως αναφέρθηκε

παραπάνω, η παραγοντοποίηση του υπολοίπου RSA *n* είναι πολύ δύσκολη, σε περίπτωση που κάτι τέτοιο επιτευχθεί όλα τα μηνύματα που κρυπτογραφούνται με το δημόσιο κλειδί θα μπορούν να αποκρυπτογραφηθούν.

- *Επίθεση επανάληψης (cycle attack)*: Σε αυτό το είδος επίθεσης το κρυπτοκείμενο αποκρυπτογραφείται επαναλαμβανόμενα μέχρι να προκύψει το αρχικό κείμενο. Ένας μεγάλος αριθμός επαναλήψεων μπορεί να καταφέρει να αποκρυπτογραφήσει οποιοδήποτε κρυπτοκείμενο. Παρόλα αυτά, η μέθοδος αυτή είναι εξαιρετικά αργή και στην περίπτωση που το μήκος του κλειδιού είναι μεγάλο, μη πρακτική.
- *Επίθεση στο υπόλοιπο RSA, n*: Η επίθεση στο υπόλοιπο *n* μπορεί να εφαρμοσθεί σε περιπτώσεις όπου υπάρχει μια ομάδα επικοινωνούντων που έχουν κλειδιά των οποίων το *n* είναι ίδιο. Όταν γίνεται χρήση του RSA είναι απαραίτητο κάθε οντότητα να διαλέγει το δικό της υπόλοιπο RSA *n*. Κάποιες φορές προτείνεται μια έμπιστη κεντρική αρχή για να διαλέγει το υπόλοιπο RSA *n*. Η ίδια αρχή στη συνέχεια διανέμει σε κάθε χρήστη του δικτύου ένα ζεύγος (e_i, d_i). Όμως, μπορεί να αποδειχθεί ότι η γνώση οποιουδήποτε ζεύγους (e_i, d_i) επιτρέπει την παραγοντοποίηση του *n.* Έτσι, οποιαδήποτε οντότητα θα μπορούσε να υπολογίσει το *d* για οποιοδήποτε άλλη οντότητα μέσα στο δίκτυο. Συνεπώς, αν ένα κρυπτογραφημένο μήνυμα στελνόταν σε μία ή παραπάνω οντότητες μέσα στο δίκτυο, υπάρχει τεχνική με την οποία ένας εισβολέας έχει μεγάλες πιθανότητες να ανακτήσει το αρχικό μήνυμα χρησιμοποιώντας μόνο μια πληροφορία που είναι δημόσια διαθέσιμη (δηλαδή το δημόσιο κλειδί (e,n)) .

Παρ' όλες τις οποιεσδήποτε αδυναμίες του, ο RSA συνεχίζει να θεωρείται ως το de facto δεδομένο για την κρυπτογράφηση δημοσίου κλειδιού, ιδιαίτερα όταν πρόκειται για δεδομένα που μεταφέρονται στο διαδίκτυο [8][20].

3.4 Εγκαθίδρυση κλειδιού με DH

Ο DHKE (Diffie-Hellman key exchange) που προτάθηκε το 1976 από τους Whitfield Diffie και Martin Hellman, ήταν το πρώτο ασύμμετρο σύστημα που δημοσιεύτηκε στην ανοιχτή βιβλιογραφία. Οι δύο εφευρέτες επηρεάστηκαν επίσης από την εργασία του Ralph Merkle. Το πρωτόκολλο παρέχει μία πρακτική λύση στο πρόβλημα διανομής κλειδιού, δηλαδή, δίνει τη δυνατότητα σε δύο μέρη να παράγουν ένα κοινό μυστικό κλειδί, επικοινωνώντας δια μέσου ενός μη ασφαλούς καναλιού. Ο DHKE αποτελεί μία εντυπωσιακή εφαρμογή του προβλήματος του διακριτού λογαρίθμου. Η θεμελιώδης αυτή τεχνική συμφωνίας κλειδιού υλοποιείται σε πολλά ανοιχτά και εμπορικά κρυπτογραφικά πρωτόκολλα όπως, στο SSH (Secure Shell), στο TLS (Transport Layer Security) και στο IPSec (Internet Protocol Security) το οποίο και θα δούμε παρακάτω. Η βασική ιδέα πίσω από τον DHKE είναι ότι η εκθετικοποίηση στο $\mathbb{Z}_p^*$, όπου *p* πρώτος, είναι μία μονόδρομη διαδικασία και ότι η εκθετικοποίηση αυτή είναι αντιμεταθετική, δηλαδή $k = (a^x)^y \equiv (a^y)^x \bmod p$. Η τιμή $k = (a^x)^y \equiv (a^y)^x \bmod p$ είναι το κοινό μυστικό που μπορεί να χρησιμοποιηθεί σαν κλειδί συνόδου μεταξύ των δύο μερών.

Ας θεωρήσουμε τώρα ότι το DHKE πρωτόκολλο λειτουργεί πάνω στο $\mathbb{Z}_p^*$. Σε αυτό το πρωτόκολλο έχουμε δύο μέρη, τη Μαρία και τον Πέτρο, που θα ήθελαν να εγκαθιδρύσουν ένα κοινόχρηστο μυστικό κλειδί. Υπάρχει πιθανώς ένα τρίτο μέρος που επιλέγει κατάλληλα τις δημόσιες παραμέτρους που απαιτούνται για την ανταλλαγή κλειδιού. Ωστόσο είναι πιθανό ότι η Μαρία και ο Πέτρος παράγουν μόνοι τους τις δημόσιες παραμέτρους. Αυστηρά μιλώντας, το DHKE αποτελείται από δύο πρωτόκολλα, το πρωτόκολλο δημιουργίας (*set up protocol*) και το κύριο πρωτόκολλο (*main protocol*) που εκτελεί την πραγματική ανταλλαγή κλειδιού. Το *set up protocol* αποτελείται από τα ακόλουθα βήματα:

1. Επιλέγεται ένας πολύ μεγάλος πρώτος αριθμός *p.*
2. Επιλέγεται ένας ακέραιος $a \in \{2,3,\ldots, p-2\}$.
3. Δημοσιεύεται ο *p* και ο *a*.

Αυτές οι δύο τιμές μερικές φορές αναφέρονται και ως παράμετροι πεδίου ορισμού (*domain parameters*). Αν η Μαρία και ο Πέτρος γνωρίζουν και οι δύο τις δημόσιες παραμέτρους *p* και *a*

που υπολογίστηκαν στη φάση της δημιουργίας, μπορούν και οι δύο να παράγουν ένα κοινό μυστικό κλειδί *k* με το πρωτόκολλο ανταλλαγής κλειδιού που φαίνεται στην Εικόνα 18.

Ακολουθεί η απόδειξη που δείχνει ότι το πρωτόκολλο είναι σωστό, δηλαδή, ότι η Μαρία και ο Πέτρος στη πραγματικότητα υπολογίζουν το ίδιο κλειδί συνόδου k_{AB}. Η Μαρία υπολογίζει $B^a \equiv (a^b)^a \equiv a^{ab} \bmod p$, ενώ ο Πέτρος υπολογίζει $A^b \equiv (a^a)^b \equiv a^{ab} \bmod p$, και με αυτό τον τρόπο και οι δύο μοιράζονται το κλειδί συνόδου $k_{AB} \equiv a^{ab} \bmod p$.

Μαρία	**Πέτρος**
Επιλέγει $a = k_{pr,A} \in \{2,\ldots,p-2\}$	Επιλέγει $b = k_{pr,B} \in \{2,\ldots,p-2\}$
Υπολογίζει $A = k_{pub,A} \equiv a^a \bmod p$	Υπολογίζει $B = k_{pub,B} \equiv a^b \bmod p$
$k_{pub,A} = A \longrightarrow$	
$\longleftarrow k_{pub,B} = B$	
$k_{AB} = k_{pub,B}^{k_{pr,A}} \equiv B^a \bmod p$	$k_{AB} = k_{pub,A}^{k_{pr,B}} \equiv A^b \bmod p$

Εικόνα 18. Diffie-Hellman Key Exchange – DHKE[6].

Το κλειδί μπορεί τώρα να χρησιμοποιηθεί για την εγκαθίδρυση μίας ασφαλούς επικοινωνίας μεταξύ της Μαρίας και του Πέτρου, π.χ., χρησιμοποιώντας το k_{AB} σαν κλειδί για έναν συμμετρικό αλγόριθμο όπως ο AES ή ο 3DES.

Παράδειγμα 14. Έστω ότι οι παράμετροι του πεδίου ορισμού είναι το *p = 29* και το *a = 2*. Το πρωτόκολλο λειτουργεί όπως παρακάτω:

Μαρία	**Πέτρος**
Επιλέγει $a = k_{pr,A} = 5$	Επιλέγει $b = k_{pr,B} = 12$
$A = k_{pub,A} = 2^5 \equiv 3 \bmod 29$	$B = k_{pub,B} = 2^{12} \equiv 7 \bmod 29$
$A = 3 \longrightarrow$	
$\longleftarrow B = 7$	
$k_{AB} = B^a = 7^5 \equiv 16 \bmod 29$	$k_{AB} = A^b = 3^{12} \equiv 16 \bmod 29$

Εικόνα 19. Παράδειγμα εκτέλεσης στο DHKE[6].

Όπως μπορούμε να δούμε, και τα δύο μέρη υπολογίζουν την τιμή $k_{AB} = 16$ η οποία μπορεί να χρησιμοποιηθεί ως κοινό μυστικό, π.χ., σαν κλειδί συνόδου για συμμετρική κρυπτογράφηση.

Οι υπολογιστικές πτυχές του DHKE είναι αρκετά παρόμοιες με αυτές του RSA. Κατά τη διάρκεια της φάσης του *set up,* παράγεται ο *p* χρησιμοποιώντας πιθανολογικούς αλγορίθμους εύρεσης πρώτων αριθμών. Ο *p* πρέπει να έχει παρόμοιο μήκος με τον RSA υπολοίπου *n*, δηλαδή, 1024 bit ή παραπάνω, έτσι ώστε να μπορεί να παρέχει ισχυρή ασφάλεια. Ο ακέραιος *a* χρειάζεται να έχει μία ειδική ιδιότητα: Πρέπει να είναι ένα πρωταρχικό στοιχείο. Το κλειδί συνόδου k_{AB} που υπολογίζεται στο πρωτόκολλο έχει το ίδιο μήκος bit με το *p*. Αν θέλαμε να το χρησιμοποιήσουμε σαν ένα συμμετρικό κλειδί για αλγορίθμους όπως ο AES, μπορούμε απλά να πάρουμε τα 128 πιο σημαντικά bit. Εναλλακτικά, μία συνάρτηση κατακερματισμού εφαρμόζεται μερικές φορές στο k_{AB}, και η έξοδος αυτής χρησιμοποιείται σαν συμμετρικό κλειδί.

Κατά τη διάρκεια του πραγματικού πρωτοκόλλου, στην αρχή πρέπει να επιλέξουμε τα μυστικά κλειδιά *a* και *b*. Αυτά πρέπει να δημιουργούνται από ένα πραγματικά τυχαίο γεννήτορα για να εμποδίζουν έναν επιτιθέμενο να μπορεί να τα ανακαλύψει. Για τον υπολογισμό των δημοσίων κλειδιών *A* και *B*, όπως και στον υπολογισμό του κλειδιού συνόδου, και οι δύο

πλευρές μπορούν να κάνουν χρήση του αλγορίθμου square-and-multiply (Αλγ. 1.4). Τα δημόσια κλειδιά τυπικά είναι προϋπολογισμένα. Η κύρια λειτουργία υπολογισμού που χρειάζεται να γίνει για την ανταλλαγή κλειδιού είναι η εκθετικοποίηση για το κλειδί συνόδου. Γενικά, επειδή τα μήκη των bit και οι υπολογισμοί του RSA και του DHKE είναι παρόμοια, απαιτούν περίπου την ίδια προσπάθεια-εργασία.

Αυτό που δείξαμε μέχρι τώρα είναι το παραδοσιακό πρωτόκολλο ανταλλαγής κλειδιού Diffie-Hellman στην ομάδα $\mathbb{Z}_p^*$, όπου το *p* είναι ένας πρώτος αριθμός. Το πρωτόκολλο όμως μπορεί να γενικευθεί, και συγκεκριμένα σε ομάδες των ελλειπτικών καμπυλών. Αυτό δίνει ώθηση στην κρυπτογραφία ελλειπτικών καμπυλών, η οποία έχει γίνει στη πράξη ένα πολύ δημοφιλές ασύμμετρο σύστημα. Για να καταλάβετε καλύτερα τις ελλειπτικές καμπύλες και τα συστήματα κρυπτογράφησης όπως αυτό του Elgamal, το οποίο σχετίζεται στενά με το DHKE, θα πρέπει να μελετήσετε το πρόβλημα του διακριτού λογαρίθμου (Ενότητα 3.4.1). Το πρόβλημα αυτό αποτελεί τη βάση του DHKE.

3.5 Παραλλαγές DH

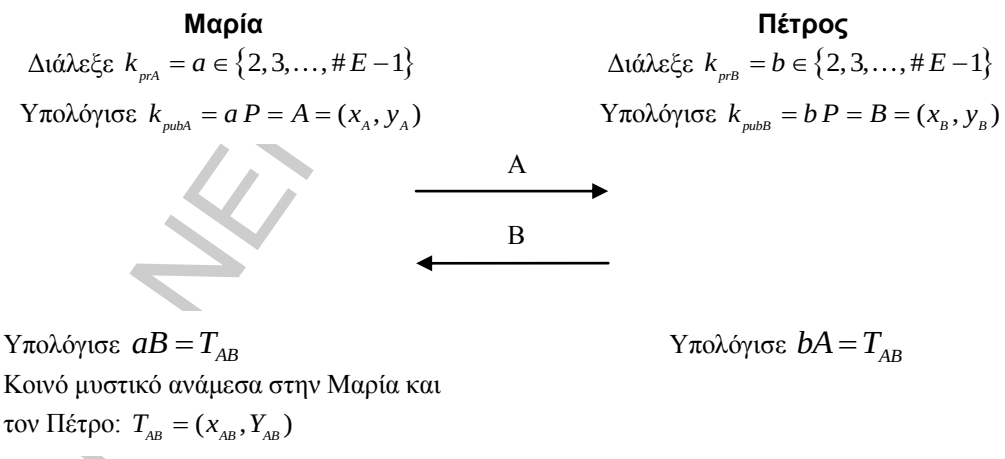
3.5.1 Ο Diffie-Hellman Ανταλλαγής Κλειδιού με Ελλειπτικές Καμπύλες

Σε πλήρη αναλογία με τον παραδοσιακό DHKE που αναφέραμε προηγουμένως, μπορούμε να πραγματοποιήσουμε ανταλλαγή κλειδιού χρησιμοποιώντας ελλειπτικές καμπύλες. Αυτό αναφέρεται ως ελλειπτικής καμπύλης ανταλλαγή κλειδιού Diffie-Hellman ή ECDH. Αρχικά θα πρέπει να συμφωνήσουμε στο πεδίο των παραμέτρων, που είναι μία κατάλληλη ελλειπτική καμπύλη πάνω στην οποία μπορούμε να δουλέψουμε και ένα πρωταρχικό στοιχείο σε αυτήν την καμπύλη.

1. Επιλέγουμε έναν πρώτο *p* και την ελλειπτική καμπύλη $E: y^2 \equiv x^3 + a \cdot x + b \bmod p$.
2. Επιλέγουμε ένα πρωταρχικό στοιχείο $P = (x_P, x_P)$.

Ο πρώτος *p*, η καμπύλη που δίνεται από τους συντελεστές *a, b* και το πρωταρχικό στοιχείο *P* αποτελούν τους παραμέτρους του πεδίου ορισμού.

Σημειώστε ότι στη πράξη η εύρεση μίας κατάλληλης ελλειπτικής καμπύλης είναι μία σχετικά δύσκολη διαδικασία. Οι καμπύλες πρέπει να έχουν ορισμένες ιδιότητες για να μπορούν να είναι ασφαλής. Η πραγματική ανταλλαγή κλειδιού γίνεται με τον ίδιο τρόπο με το παραδοσιακό πρωτόκολλο του DH.



Εικόνα 20. Ο DH Ανταλλαγής Κλειδιού με Ελλειπτικές Καμπύλες – ECDH[11].

Η ορθότητα του πρωτοκόλλου είναι εύκολη να αποδειχθεί. Η Μαρία υπολογίζει το $aB = a(bP)$ ενώ ο Πέτρος υπολογίζει το $bA = b(aP)$. Επειδή η πρόσθεση σημείου είναι προσεταιριστική, και τα δύο μέρη υπολογίζουν το ίδιο αποτέλεσμα, δηλαδή το σημείο $T_{AB} = abP$.

Όπως μπορούμε να δούμε στο πρωτόκολλο, η Μαρία και ο Πέτρος επιλέγουν τα μυστικά κλειδιά *a* και *b*, αντίστοιχα, τα οποία είναι και τα δύο μεγάλοι ακέραιοι. Με τα μυστικά κλειδιά και οι δύο παράγουν τα αντίστοιχα δημόσια κλειδιά *A* και *B*, τα οποία είναι σημεία στη καμπύλη. Τα δημόσια κλειδιά υπολογίζονται με πολλαπλασιασμό σημείου. Τα δύο μέρη ανταλλάσουν αυτούς τους δημόσιους παραμέτρους ο ένας με τον άλλον. Το κοινό μυστικό T_{AB} υπολογίζεται στη συνέχεια από την Μαρία και τον Πέτρο εκτελώντας ένα δεύτερο πολλαπλασιασμό σημείου περιλαμβάνοντας το δημόσιο κλειδί που έλαβαν και τη δική τους μυστική παράμετρο. Το κοινό μυστικό T_{AB} μπορεί να χρησιμοποιηθεί για τη παραγωγή ενός κλειδιού συνόδου, π.χ., σαν είσοδο για τον αλγόριθμο AES. Σημειώστε ότι οι δύο συντεταγμένες (X_{AB}, Y_{AB}) δεν είναι ανεξάρτητες η μία από την άλλη: Δοθέντος x_{AB}, η άλλη συντεταγμένη μπορεί να υπολογιστεί απλά με την εισαγωγή της τιμής *x* στην εξίσωση της ελλειπτικής καμπύλης. Έτσι, μόνο μία από τις δύο συντεταγμένες θα πρέπει να χρησιμοποιηθεί για την παραγωγή ενός κλειδιού συνόδου. Ας δούμε στη συνέχεια ένα παράδειγμα με μικρούς αριθμούς:

Παράδειγμα 15. Υποθέτουμε τον ECDH με τους παρακάτω παραμέτρους πεδίου ορισμού. Η ελλειπτική καμπύλη είναι η $y^2 \equiv x^3 + 2x + 2 \bmod 17$, η οποία σχηματίζει μία κυκλική ομάδα τάξης *#E = 19*. Το βασικό σημείο είναι το *P = (5,1)*. Το πρωτόκολλο ενεργεί ως ακολούθως:

Μαρία	**Πέτρος**
Διάλεξε $a = k_{pr,A} = 3$	Διάλεξε $b = k_{pr,B} = 10$
Υπολόγισε $A = k_{pub,A} = 3P = (10,6)$	Υπολόγισε $B = k_{pub,B} = 10P = (7,11)$
	A →
	← B
$T_{AB} = aB = 3(7,11) = (13,10)$	$T_{AB} = bA = 10(10,6) = (13,10)$

Εικόνα 21. Παράδειγμα εκτέλεσης πρωτοκόλλου ECDH[3].

Μία από τις συντεταγμένες του κοινού μυστικού T_{AB} μπορεί να χρησιμοποιηθεί σαν κλειδί συνόδου. Στην πράξη, συχνά η συντεταγμένη *x* είναι κατακερματισμένη και στη συνέχεια χρησιμοποιείται σαν συμμετρικό κλειδί. Τυπικά, δε χρειάζονται όλα τα bit. Για παράδειγμα, σε ένα 160-bit σύστημα ECC, ο κατακερματισμός της συντεταγμένης *x* με τον αλγόριθμο κατακερματισμού SHA-1 έχει σαν αποτέλεσμα έξοδο μεγέθους 160 bit, από τα οποία μόνο τα 128 θα χρησιμοποιηθούν σαν ένα κλειδί AES.

Σημειώστε ότι οι ελλειπτικές καμπύλες δεν περιορίζονται στον DHKE. Πραγματικά, σχεδόν όλα τα πρωτόκολλα διακριτού λογαρίθμου, ειδικότερα ψηφιακών υπογραφών και κρυπτογράφησης, μπορούν να υλοποιηθούν [3][6][10].

3.5.2 Σύγκριση του DH με τον ECDH

Οι πρώτες εκδόσεις του μηχανισμού DH ήταν ευάλωτες σε επιθέσεις *man-in-the-middle*. Σε αυτή την επίθεση ο χρήστης *C* παρεμβάλλεται στην επικοινωνία των *A* και *B* και όταν ανταλλάσσουν τις δημόσιες τιμές τους τις αντικαθιστά με τις δικές του. Δηλαδή όταν ο *A* μεταδίδει την δημόσια τιμή του στον *B*, ο *C* την αντικαθιστά με την δικιά του και την στέλνει στον *B*. Ομοίως όταν ο *B* στέλνει την δημόσια τιμή του στον *A*. Σαν συνέπεια, οι *C* και *A* συμφωνούν για ένα μυστικό κλειδί και οι *C* και *B* συμφωνούν για ένα άλλο κλειδί. Έτσι ο *C* μπορεί να διαβάσει τα μηνύματα που μεταδίδουν ο *A* στον *B* και πιθανώς να τα τροποποιήσει πριν τα προωθήσει σε έναν από τους δύο.

Για την αντιμετώπιση του *man-in-the-middle,* η απλούστερη λύση είναι η χρήση δημόσιων κλειδιών, τα οποία δε μεταφέρονται μέσω του καναλιού και δεν είναι εφήμερα. Ένας άλλος τρόπος είναι η χρήση ψηφιακών υπογραφών. Τα μηνύματα ανταλλάσσονται υπογεγραμμένα με τα ιδιωτικά κλειδιά των *A* και *B*, ενώ χρησιμοποιούνται και πιστοποιητικά για την απόκτηση των σωστών δημοσίων κλειδιών. Έτσι, ακόμα και αν κάποιος είναι σε θέση να παρακολουθεί την

επικοινωνία των *A* και *B*, δεν μπορεί να πλαστογραφήσει τα μηνύματα. Είτε με τη χρήση ελλειπτικών καμπυλών, είτε με τη χρήση πρώτων αριθμών, ο αλγόριθμος DH θεωρείται ασφαλής και χρησιμοποιείται για την ανταλλαγή κλειδιών.

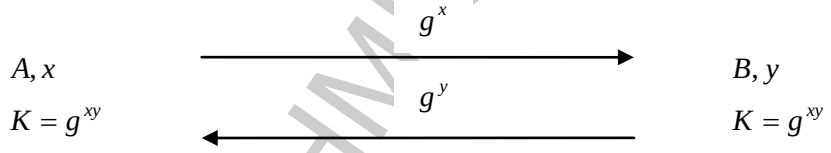
Ο ECDH έχει πλεονέκτημα ταχύτητας σε σχέση με τον απλό DH, καθώς εξαλείφει την ανάγκη για εκθετικοποίηση κατά modulo. Το μέγεθος του κλειδιού που απαιτείται για τον ECDH είναι πολύ μικρότερο από αυτό του απλού DH, για το ίδιο επίπεδο ασφάλειας. Από την άλλη πλευρά, η υποκείμενη θεωρία για την αντίληψη και χρήστη των ελλειπτικών καμπύλων είναι πιο δύσκολη. Για το λόγο αυτό, υπάρχει διστακτικότητα στην χρήση κρυπτογραφικών συστημάτων που βασίζονται στις ελλειπτικές καμπύλες.

3.5.3 Εφήμερο DH

Στο εφήμερο DH οι δύο οντότητες εκτελούν τα ακόλουθα βήματα:

1. Ο *A* επιλέγει $x \in_R [1, q-1]$ και στέλνει το $R_A = g^x$ στον *B*.
2. Ο *B* επιλέγει $y \in_R [1, q-1]$ και στέλνει το $R_B = g^y$ στον *A*.
3. Ο *A* υπολογίζει το $K = (R_B)^x = g^{xy}$.
4. $K = (R_A)^y = g^{xy}$.

Ενώ το εφήμερο πρωτόκολλο του DH παρέχει *έμμεση πιστοποίηση κλειδιού* στην παρουσία παθητικών παρατηρητών, δεν παρέχει όμως από μόνου του καμία χρήσιμη υπηρεσία στην παρουσία ενεργητικών παρατηρητών, αφού δεν παρέχει σε κάποια οντότητα οποιεσδήποτε εγγυήσεις σχετικά με τη ταυτότητα της οντότητας που επικοινωνεί η πρώτη. Αυτό το μειονέκτημα μπορεί να ξεπεραστεί χρησιμοποιώντας δημόσια κλειδιά που έχουν εκδοθεί από μίαέμπιστη ΑΠ [3].

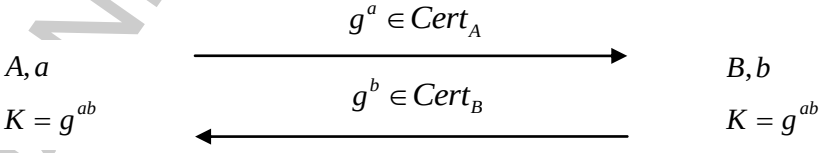


Εικόνα 22. Εφήμερο DH[3].

3.5.4 Στατικό DH

Στο στατικό DH οι δύο οντότητες εκτελούν τα ακόλουθα βήματα:

1. Ο *A* στέλνει ένα $Cert_A$ στον *B*.
2. Ο *B* στέλνει ένα $Cert_B$ στον *A*.
3. Ο *A* υπολογίζει το $K = (Y_B)^a = g^{ab}$.
4. Ο *B* υπολογίζει $K = (Y_A)^b = g^{ab}$.



Εικόνα 23. Στατικό DH[3].

Επειδή κάθε οντότητα διασφαλίζεται ότι κατέχει ένα αυθεντικό αντίγραφο του δημόσιου κλειδιού άλλης οντότητας, το στατικό πρωτόκολλο DH προσφέρει *έμμεση πιστοποίηση κλειδιού*. Ένα κύριο μειονέκτημα, ωστόσο, είναι ότι ο *A* και *B* υπολογίζουν το ίδιο κοινόχρηστο μυστικό κλειδί $K = g^{ab}$ για κάθε εκτέλεση του πρωτοκόλλου.

Τα μειονεκτήματα του εφήμερου και του στατικού πρωτοκόλλου DH μπορούν να αμβλυνθούν χρησιμοποιώντας κρυπτογραφικό υλικό στο σχηματισμό των κοινόχρηστων

μυστικών κλειδιών. Ένα παράδειγμα ενός πρώιμου πρωτοκόλλου που σχεδιάστηκε με αυτόν τον τρόπο είναι το MTI/C0 [3].

4. Μηχανισμός συμφωνίας κλειδιού σε γνωστά πρωτόκολλα ασφαλείας

4.1 Οι Στόχοι της Συμφωνίας Κλειδιού

Η ενότητα αυτή αναφέρεται με λεπτομέρειες στους στόχους των ασύμμετρων πρωτοκόλλων εγκαθίδρυσης επικυρωμένων κλειδιών. Η πολυπλοκότητα και η ποικιλία αυτών των στόχων εξηγεί εν μέρει τις δυσκολίες που εμπλέκονται στη σχεδίαση ασφαλών πρωτοκόλλων.

Ο βασικός στόχος οποιουδήποτε πρωτοκόλλου εγκαθίδρυσης επικυρωμένου κλειδιού είναι να διανέμει δεδομένα κλειδιού. Ιδανικά, το εγκατεστημένο κλειδί θα πρέπει να έχει ακριβώς τις ίδιες ιδιότητες σαν ένα κλειδί που εγκαταστάθηκε πρόσωπο-με-πρόσωπο. Για παράδειγμα θα πρέπει να μοιράζονται από τις (δύο) καθορισμένες οντότητες, θα πρέπει να κατανέμονται ομοιόμορφα στη τύχη από το χώρο που παράγεται το κλειδί και καμία μη εξουσιοδοτημένη οντότητα δεν πρέπει να μαθαίνει κάτι για το κλειδί. Ένα πρωτόκολλο που κατορθώνει αυτό τον ιδεαλιστικό στόχο, μπορεί στη συνέχεια να χρησιμοποιηθεί σαν αντικατάσταση της εγκαθίδρυσης κλειδιού μέσω δύο συναλλασσόμενων φυσικών προσώπων.

Δυστυχώς, ένας τέτοιος αφηρημένος στόχος δεν επιτυγχάνεται εύκολα και δεν είναι καθόλου εύκολη εργασία να προσδιορίσεις και να διατυπώσεις με ακρίβεια τις απαιτήσεις ασφάλειας της επικυρωμένης εγκαθίδρυσης κλειδιού. Παρ 'όλα αυτά τα χρόνια αρκετές συγκεκριμένες ιδιότητες ασφάλειας και επίδοσης αναγνωρίστηκαν σαν επιθυμητές. Το πρώτο βήμα είναι να προσδιορίσεις τι τύπου επιθέσεων είναι ζωτικής σημασίας για ένα πρωτόκολλο να αντέχει. Από τη στιγμή που τα πρωτόκολλα χρησιμοποιούνται σε ανοιχτά δίκτυα όπως το διαδίκτυο, ένα ασφαλές πρωτόκολλο θα πρέπει να αντέχει και στις παθητικές επιθέσεις (όπου ένας παρατηρητής προσπαθεί να εμποδίσει το πρωτόκολλο να ολοκληρώσει τους στόχους του παρατηρώντας απλά τις έντιμες οντότητες να εκτελούν το πρωτόκολλο) και στις ενεργητικές επιθέσεις (όπου ένας παρατηρητής υπονομεύει επιπλέον τις επικοινωνίες εγχέοντας, διαγράφοντας, τροποποιώντας ή επαναλαμβάνοντας μηνύματα). Το δεύτερο βήμα είναι να προσδιορίσεις τους συγκεκριμένους στόχους ασφάλειας που είναι ζωτικής σημασίας να παρέχει ένα πρωτόκολλο. Οι θεμελιώδεις στόχοι ασφάλειας που περιγράφονται παρακάτω θεωρούνται ζωτικής σημασίας σε κάθε εφαρμογή.

Ας υποθέσουμε ότι ο *A* και ο *B* είναι δύο έντιμες οντότητες, δηλαδή, θεμιτές οντότητες που εκτελούν τα βήματα ενός πρωτοκόλλου σωστά.

1. *Έμμεση πιστοποίηση κλειδιού*: Ένα πρωτόκολλο συμφωνίας κλειδιού φέρεται να παρέχει έμμεση πιστοποίηση κλειδιού (του *B* στον *A*) αν η οντότητα *A* διασφαλίζει ότι καμία άλλη οντότητα πέραν από την ειδικά αναγνωρισμένη οντότητα *B* μπορεί να μάθει τη τιμή ενός συγκεκριμένου μυστικού κλειδιού. Σημειώστε ότι η παραπάνω ιδιότητα δε σημαίνει απαραίτητα ότι ο *A* διασφαλίζει τον *B* ότι πραγματικά κατέχει το κλειδί.
2. *Ρητή πιστοποίηση κλειδιού*: Ένα πρωτόκολλο συμφωνίας κλειδιού φέρεται να παρέχει ρητή πιστοποίηση κλειδιού του *B* στον *A*) αν η οντότητα *A* εξασφαλίζει ότι η δεύτερη οντότητα *B* έχει πράγματι υπολογίσει το συμφωνημένο κλειδί. Το πρωτόκολλο παρέχει *έμμεση επιβεβαίωση κλειδιού* αν ο *A* εξασφαλίζει ότι ο *B* μπορεί να υπολογίσει το συμφωνημένο κλειδί. Ενώ η ρητή επιβεβαίωση κλειδιού φαίνεται να παρέχει ισχυρότερες εγγυήσεις στον *A* από ότι η έμμεση επιβεβαίωση κλειδιού, φαίνεται ότι, για όλους τους πρακτικούς σκοπούς, οι εγγυήσεις είναι στην πράξη ίδιες. Δηλαδή, η εγγύηση που απαιτεί ο *A*, στη πράξη είναι απλώς ότι ο *B* μπορεί να υπολογίσει το κλειδί και όχι ότι ο *B* έχει πραγματικά υπολογίσει το κλειδί. Όντως στη πράξη, ακόμη και αν ένα πρωτόκολλο παρέχει ρητή επιβεβαίωση κλειδιού, δε μπορεί να εγγυηθεί στον *A* ότι ο *B* δε θα χάσει το κλειδί ανάμεσα στην εγκαθίδρυση και στη χρήση του κλειδιού.

Από μόνη της η επιβεβαίωση κλειδιού δεν είναι μία και τόσο χρήσιμη υπηρεσία, αλλά είναι επιθυμητή μόνο όταν συνοδεύεται με έμμεση πιστοποίηση κλειδιού. Ένα πρωτόκολλο συμφωνίας κλειδιού φέρεται να παρέχει ρητή πιστοποίηση κλειδιού (του *B* στον *A*) αν παρέχονται μαζί και η έμμεση πιστοποίηση κλειδιού και η επιβεβαίωση κλειδιού (του *B* στον *A*). Ένα πρωτόκολλο συμφωνίας κλειδιού που παρέχει έμμεση πιστοποίηση κλειδιού και στις δύο συμμετέχουσες οντότητες ονομάζεται πρωτόκολλο *συμφωνίας πιστοποιημένου κλειδιού (authenticated key agreement - AK)*, ενώ εκείνο που παρέχει ρητή πιστοποίηση κλειδιού και

στις δύο συμμετέχουσες οντότητες ονομάζεται πρωτόκολλο *συμφωνίας πιστοποιημένου κλειδιού με επιβεβαίωση κλειδιού (authenticated key agreement with key confirmation - AKC)*.

Υπάρχουν μερικές ακόμη επιθυμητές ιδιότητες ασφάλειας που έχουν προσδιοριστεί. Τυπικά, η σημαντικότητα της παροχής αυτών των ιδιοτήτων εξαρτάται κάθε φορά από την εφαρμογή. Έστω ότι για τα παρακάτω ο *A* και ο *B* είναι δύο έντιμες οντότητες.

1. *Ασφάλεια γνωστού κλειδιού*: Κάθε εκτέλεση ενός πρωτοκόλλου συμφωνίας κλειδιού μεταξύ του *A* και του *B* θα πρέπει να παράγει ένα μοναδικό μυστικό κλειδί, τέτοια κλειδιά ονομάζονται κλειδιά συνόδου (session keys). Τα κλειδιά συνόδου είναι σκόπιμα για να περιορίζουν τον όγκο των δεδομένων που είναι διαθέσιμα κάθε φορά για κρυπτανάλυση. Ένα πρωτόκολλο θα πρέπει να επιτυγχάνει αυτό το στόχο απέναντι σε έναν τρίτο παρατηρητή που έχει μάθει κάποια άλλα κλειδιά συνόδου.
2. *Απόρρητο προς τα εμπρός*: Μακροπρόθεσμα τα μυστικά κλειδιά μίας ή περισσοτέρων οντοτήτων βρίσκονται σε κίνδυνο, η μυστικότητα των προηγούμενων κλειδιών συνόδου που εγκαθιδρύθηκαν από έντιμες οντότητες δεν επηρεάζονται. Μερικές φορές γίνεται ένας διαχωρισμός ανάμεσα στη περίπτωση που ένα μυστικό κλειδί μίας οντότητας είναι σε κίνδυνο (*half forward secrecy*) και στη περίπτωση που τα κλειδιά και των δύο οντοτήτων που συμμετέχουν είναι σε κίνδυνο (*full forward secrecy*).
3. *Κίνδυνος πλαστογραφίας κλειδιού*: Ας υποθέσουμε ότι μακροπρόθεσμα φανερώνεται το μυστικό κλειδί του *A*. Σαφώς ένας παρατηρητής που γνωρίζει τη τιμή αυτή μπορεί να υποδυθεί τον *A*, αφού ακριβώς η τιμή αυτή προσδιορίζει τον *A*. Ωστόσο, μπορεί να είναι επιθυμητό σε μερικές περιπτώσεις η απώλεια αυτή να μην επιτρέπει στον παρατηρητή να υποδυθεί άλλες οντότητες στον *A*.
4. *Άγνωστο μοίρασμα κλειδιού*: Η οντότητα *B* δε μπορεί να εξαναγκαστεί να μοιραστεί ένα κλειδί με την οντότητα *A* χωρίς να το γνωρίζει, δηλαδή, όταν ο *B* πιστεύει ότι το κλειδί μοιράζεται με κάποια οντότητα $C \neq A$, και ο *A* (σωστά) πιστεύει ότι το κλειδί μοιράζεται με τον *B*. Ένα υποθετικό σενάριο όπου μία επίθεση άγνωστου μοιράσματος κλειδιού μπορεί να έχει καταστροφικές συνέπειες, παρουσιάζεται στη συνέχεια. Υποθέστε ότι ο *B* είναι ένα υποκατάστημα τράπεζας και ο *A* είναι ένας κάτοχος λογαριασμού της. Εκδίδονται πιστοποιητικά από τα κεντρικά γραφεία της τράπεζας και σε κάθε πιστοποιητικό βρίσκονται οι πληροφορίες του λογαριασμού του κατόχου. Υποθέστε ότι το πρωτόκολλο για ηλεκτρονική κατάθεση χρημάτων είναι να ανταλλάξει ένα κλειδί με ένα υποκατάστημα τράπεζας μέσω μίας αμοιβαίας συμφωνίας πιστοποιημένου κλειδιού. Μόλις ο *B* έχει πιστοποιήσει την οντότητα μετάδοσης, κρυπτογραφημένα χρήματα καταθέτονται στον αριθμό λογαριασμού στο πιστοποιητικό. Φανταστείτε τώρα ότι δε γίνεται περαιτέρω πιστοποίηση στο κρυπτογραφημένο μήνυμα της κατάθεσης (το οποίο θα μπορούσε να είναι η περίπτωση για οικονομία εύρους γραμμής μετάδοσης). Αν η επίθεση που αναφέρθηκε παραπάνω ξεκίνησε επιτυχημένα, τότε η κατάθεση θα γίνει στο λογαριασμό του *C* αντί στο λογαριασμό του *A[30,31,32]*.

4.2 Το πρωτόκολλο IPSec

Το IPSec παρέχει μία πλήρη αρχιτεκτονική πρωτοκόλλου για την πιστοποίηση και την εξασφάλιση των περιεχομένων των IP (Internet Protocol) πακέτων. Δίνει τη δυνατότητα κρυπτογράφησης και πιστοποίησης ολόκληρου του ωφέλιμου φορτίου του IP πακέτου μέσω μία ψηφιακής υπογραφής. Η αρχιτεκτονική καθορίζει:

- μία επικεφαλίδα πιστοποίησης (AH) και ένα πρωτόκολλο που χρησιμοποιείτε για να παρέχει περιεχόμενα πακέτου IP με μία ψηφιακή υπογραφή ή μία σύνοψη μηνύματος για την επιβεβαίωση της αυθεντικότητας,
- έναν ενθυλακωμένο μηχανισμό πρωτοκόλλου ασφάλειας ωφέλιμου φορτίου (ESP) για την κρυπτογράφηση των περιεχομένων των IP πακέτων (είτε IPv4 είτε IPv6),
- συσχετισμούς ασφάλειας, μία μονόδρομη end-to-end σύνδεση μεταξύ δύο σημείων που παρέχει ασφαλή μετάδοση δεδομένων. Ο end-to-end συσχετισμός ασφάλειας ταυτοποιείται μέσω ενός SPI (security parameter index), μίας IP διεύθυνσης προορισμού και ενός αναγνωριστικού πρωτοκόλλου ασφαλείας (δηλαδή, AH ή ESP). Καθορίζονται δύο τύποι συσχετισμού ασφάλειας: λειτουργία μεταφοράς (transport mode), δηλαδή, κρυπτογράφηση end-to-end και λειτουργία σήραγγας (tunnel mode),
- συνιστώμενοι αλγόριθμοι για αυθεντικοποίηση και κρυπτογράφηση,

- και το internet key exchange (IKE), δηλαδή την ασφαλή διαχείριση και διανομή των κλειδιών κρυπτογράφησης.

Με αυτό τον τρόπο το IPSec παρέχει έλεγχο πρόσβασης, αυθεντικοποίηση προέλευσης δεδομένων και κρυπτογράφηση. Μπορεί να χρησιμοποιηθεί για να προστατέψει μία η περισσότερες διαδρομές ανάμεσα στα τελικά σημεία και μπορεί να χρησιμοποιήσει διαφορετικούς συσχετισμούς ασφάλειας, δηλαδή (simplex paths), για την υποστήριξη και των δύο κατευθύνσεων της επικοινωνίας. Η αρχιτεκτονική επίσης προορίζεται για να παρέχει υποστήριξη για IP πακέτα πολλαπλών διανομών (multicast).

4.2.1 Επικεφαλίδα αυθεντικοποίησης του πρωτοκόλλου IP (authentication header - AH)

Η επικεφαλίδα αυθεντικοποίησης του πρωτοκόλλου ίντερνετ (AH) είναι μέρος της αρχιτεκτονικής του IPSec για την εξασφάλιση των περιεχομένων του πακέτου IP κατά τη διάρκεια της μετάδοσης. Η (AH) διασφαλίζει τη γνησιότητα των δεδομένων στο παραλήπτη με τη συμπερίληψη μίας τιμής ελέγχου ακεραιότητας (integrity check value - ICV). Στη πραγματικότητα αυτό είναι μία ψηφιακή υπογραφή. Αν η τιμή ICV που παραλαμβάνεται με το μήνυμα είναι σωστή, τότε υπάρχει μεγάλη βεβαιότητα ότι το περιεχόμενο του μηνύματος δεν έχει αλλοιωθεί καθοδόν. Η ορθότητα της ICV τιμής επιβεβαιώνεται μέσω ενός αλγορίθμου κρυπτογράφησης και ενός διαμοιρασμένου μυστικού κλειδιού γνωστού μόνο στα δύο μέρη της επικοινωνίας. Η επικεφαλίδα προορίζεται να εισαχθεί μετά την κεφαλίδα IP και πριν την κεφαλίδα ESP.

next header	**payload length**	reserved
security parameters index (SPI)		
sequence number		
authentication data (variable)		

Εικόνα 24. Η μορφή του πρωτοκόλλου της επικεφαλίδας αυθεντικοποίησης (AH) στο IPSec[22].

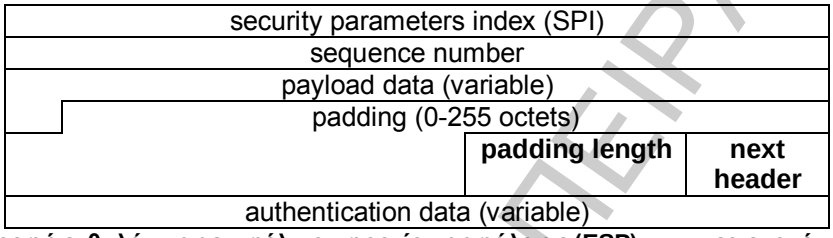
- Το πεδίο next header προσδιορίζει το επόμενο ωφέλιμο φορτίο μετά το AH. Αυτό τυπικά θα είναι το ενθυλακωμένο ωφέλιμο φορτίο ασφάλειας (ESP) ή ένα επίπεδο πρωτοκόλλου μεταφοράς ή ένα επίπεδο πρωτοκόλλου εφαρμογής και θα αναγνωρίζεται με το πρότυπο αριθμό πρωτοκόλλου IP.
- Το πεδίο payload length περιέχει μία τιμή δυαδικού αριθμού που δείχνει το μήκος της (AH). Το πεδίο reserved πρέπει να ρυθμίζεται με μηδενικές τιμές.
- Το (SPI) είναι μία κοινή 32-bit τιμή, η οποία σε συνδυασμό με την διεύθυνση IP του προορισμού και του πρωτοκόλλου ασφάλειας (σε αυτή τη περίπτωση του AH), προσδιορίζει μοναδικά το συσχετισμό ασφάλειας (SA). Ο συσχετισμός ασφάλειας είναι μία μονόδρομη (one-way) πορεία ανάμεσα στα δύο άκρα της ασφαλής σύνδεσης. Μία βάση δεδομένων συσχετισμού ασφάλειας (SAD) κατέχει το καθένα από τα μέρη που επικοινωνούν, αποθηκεύει πληροφορίες σχετικά με τον αλγόριθμο αυθεντικοποίησης και πληροφορίες κλειδιού που χρησιμοποιούνται για παραγωγή και έλεγχο της τιμής ICV που συσχετίζεται με το συσχετισμό ασφάλειας (SA).
- Το πεδίο sequence number περιέχει ένα μετρητή ο οποίος αρχικοποιείται στο 0 όταν εγκαθιδρύεται ο (SA) και αυξάνεται για κάθε επακόλουθο πακέτο. Το πεδίο authentication data περιέχει τη τιμή του (ICV). Ο υπολογισμός της τιμής ελέγχου ακεραιότητας υπολογίζεται πάνω σε ολόκληρη την IP επικεφαλίδα και τα ωφέλιμα πεδία. Κατάλληλοι αλγόριθμοι ασφάλειας/κρυπτογράφησης για την παραγωγή και τον έλεγχο της τιμής ακεραιότητας (ICV) περιλαμβάνουν συμμετρικούς κώδικες (π.χ. DES) ή μονόδρομες (δηλ. ασύμμετρες) συναρτήσεις κατακερματισμού (π.χ. MD5, SHA-1 κ.α.) [22].

4.2.2 Ενθυλάκωση ωφέλιμου φορτίου ασφάλειας (ESP)

Το ESP είναι μέρος της αρχιτεκτονικής του IPSec για τη διασφάλιση των περιεχομένων του IP πακέτου κατά τη διάρκεια της μετάδοσης. Διασφαλίζει την εμπιστευτικότητα των περιεχομένων του IP πακέτου (δηλ. των δεδομένων) κωδικοποιώντας τα σε κρυπτογραφημένη μορφή. Κατά

την παραλαβή, το ESP πρέπει να αποκρυπτογραφήσει τα δεδομένα. Ένα πλήθος από διαφορετικούς αλγόριθμους κρυπτογράφησης μπορούν να χρησιμοποιηθούν για τη κρυπτογράφηση του ίδιου. Το ESP προορίζεται για να εισαχθεί μετά την κεφαλίδα IP (και κάθε κεφαλίδα αυθεντικοποίησης, AH). Η κωδικοποίηση των διάφορων πεδίων του πρωτοκόλλου είναι όπως παρακάτω:

- Ο δείκτης ασφάλειας παραμέτρων (SPI) είναι ο ίδιος όπως και για τη κεφαλίδα αυθεντικοποίησης όπως περιγράφηκε προηγουμένως. Ο αριθμός ακολουθίας (sequence number) είναι ένα υποχρεωτικό πεδίο που περιέχει ένα μετρητή πακέτου.
- Το ωφέλιμο φορτίο δεδομένων (payload data) περιέχει δεδομένα σε μία μορφή πρωτοκόλλου όπως καθορίζεται από το πεδίο next header. Αυτό τυπικά θα είναι ένα επίπεδο μεταφοράς ή ένα επίπεδο εφαρμογής πρωτοκόλλου, και θα μπορεί να αναγνωρισθεί με το πρότυπο αριθμό του IP πρωτοκόλλου.
- Αν ο αλγόριθμος που χρησιμοποιείται για να κρυπτογραφήσει το ωφέλιμο φορτίο απαιτεί συγχρονισμό κρυπτογράφησης, τότε αυτό μπορεί να μεταφερθεί στο πεδίο του ωφέλιμου φορτίου.



Εικόνα 25. Μορφή ενθυλάκωσης ωφέλιμου φορτίου ασφάλειας (ESP) των περιεχομένων ενός πακέτου IP[23].

- Το πεδίο padding επιτρέπει το μήκος του ωφέλιμου φορτίου να ρυθμίζεται σε ένα καθορισμένο μήκος σύμφωνα με τις ανάγκες του αλγορίθμου κρυπτογράφησης.
- Το πεδίο αυθεντικοποίησης δεδομένων περιέχει μία τιμή ελέγχου ακεραιότητας (ICV) υπολογισμένο επί του ESP πακέτου μείον των δεδομένων αυθεντικοποίησης [23].

4.2.3 Internet Key Exchange (IKE)

Όπως αναφέραμε παραπάνω, συσχετίσεις ασφάλειας χρησιμοποιούνται με το IPSec για να δηλώσουν την επεξεργασία που έγινε σε ένα συγκεκριμένο πακέτο. Ένα εξερχόμενο πακέτο παράγει ένα χτύπημα στη Πολιτική Ασφαλείας Δεδομένων (Security Policy Database - SPD) η οποία εγγραφή δείχνει σε έναν ή περισσότερους συσχετισμούς ασφάλειας. Αν δεν υπάρχει SA που να δίνει υπόσταση στη πολιτική από το SPD είναι απαραίτητο να δημιουργηθεί μία. Εδώ είναι όπου το (IKE) μπαίνει στο παιχνίδι. Ολόκληρος ο σκοπός του IKE είναι να εγκαθιδρύει διαμοιρασμένες παραμέτρους ασφάλειας και πιστοποιημένα κλειδιά, με άλλα λόγια, συσχετίσεις ασφάλειας μεταξύ των IPSec μελών.

Το πρωτόκολλο IKE είναι ένα υβριδικό πρωτόκολλο των Oakley και SKEME πρωτοκόλλων και λειτουργεί μέσα σε ένα πλαίσιο που καθορίζεται από το Internet Security Association and Key Management Protocol (ISAKMP). Το ISAKMP καθορίζει μορφές πακέτου, χρονόμετρα αναμετάδοσης και κατασκευαστικές απαιτήσεις του μηνύματος. Τα Oakley και SKEME δηλώνουν τα βήματα που δύο μέλη θα πρέπει να κάνουν για να εγκαθιδρύσουν ένα κοινό αυθεντικοποιημένο κλειδί. Το IKE χρησιμοποιεί τη γλώσσα του ISAKMP για να περιγράψει αυτές, αλλά και άλλες ανταλλαγές.

Το IKE είναι στη πραγματικότητα ένα γενικής χρήσης ασφαλές πρωτόκολλο ανταλλαγής και μπορεί να χρησιμοποιηθεί για διαπραγμάτευση μίας πολιτικής και εγκαθίδρυση γνήσιου κρυπτογραφικού υλικού για διάφορες ανάγκες. Χρησιμοποιεί την έννοια του συσχετισμού ασφάλειας αλλά η φυσική δομή ενός IKE SA είναι διαφορετική από ένα IPSec SA. Το IKE SA καθορίζει τον τρόπο με τον οποίο τα δύο μέλη επικοινωνούν, για παράδειγμα, ποιον αλγόριθμο να χρησιμοποιήσουν για να κρυπτογραφήσουν τη κίνηση IKE, πώς να πιστοποιήσουν το απομακρυσμένο μέλος, κτλ. Στη συνέχεια το IKE SA χρησιμοποιείτε για να παράγει οποιοδήποτε αριθμό IPSec SAs ανάμεσα στα μέλη.

Το IPSec SA που εγκαθιδρύεται από το IKE μπορεί προαιρετικά να έχει τέλεια προς τα εμπρός μυστικότητα των κλειδιών. Περισσότερα από ένα ζευγάρι IPSec SAs μπορούν να

δημιουργηθούν αμέσως χρησιμοποιώντας απλή ανταλλαγή IKE, και οποιοσδήποτε αριθμός τέτοιων ανταλλαγών μπορούν να εκτελεστούν από ένα απλό IKE SA. Αυτός ο πλούτος των επιλογών κάνει το IKE πολύ επεκτάσιμο αλλά και πολύ πολύπλοκο.

Το πρωτόκολλο IKE εκτελείτε από κάθε μέρος που θα εκτελεί IPSec. Το μέλος IKE είναι επίσης και IPSec μέλος. Με άλλα λόγια, για να δημιουργηθούν IPSec συσχετισμοί με μία απομακρυσμένη οντότητα, μιλάς με IKE με αυτή την οντότητα και όχι με κάποια διαφορετική. Το πρωτόκολλο είναι τύπου απαίτησης-απάντησης με έναν αρχικοποιητή και έναν ανταποκριτή. Ο αρχικοποιητής είναι το μέρος που εισάγεται από το IPSec για να εγκαθιδρύσει μερικούς συσχετισμούς ασφάλειας σαν αποτέλεσμα ενός εξερχόμενου πακέτου που ταιριάζει μία εγγραφή SPD. Αυτό αρχικοποιεί το πρωτόκολλο στον ανταποκριτή.

Το SPD του IPSec χρησιμοποιείται για να αναθέσει στο IKE τι να εγκαθιδρύσει αλλά όχι όμως και πώς να το κάνει. Πως το IKE εγκαθιδρύει τους συσχετισμούς ασφάλειας IPSec βασίζεται στις δικές του ρυθμίσεις πολιτικής. Το IKE καθορίζει πολιτική με όρους προστασίας ολόκληρων σουιτών. Κάθε σουίτα προστασίας πρέπει τουλάχιστον να δηλώνει τον αλγόριθμο κρυπτογράφησης, τον αλγόριθμο κατακερματισμού, την ομάδα Diffie-Hellman και τη μέθοδο της πιστοποιημένης χρήσης. Η πολιτική της IKE βάσης δεδομένων είναι η λίστα όλων των σουιτών προστασίας που σταθμίζονται κατά σειρά προτίμησης. Η πρώτη διαπραγμάτευση που κάνουν τα δύο μέλη του IKE είναι να συμφωνήσουν στη συγκεκριμένη σουίτα πολιτικής που θα εφαρμόσουν.

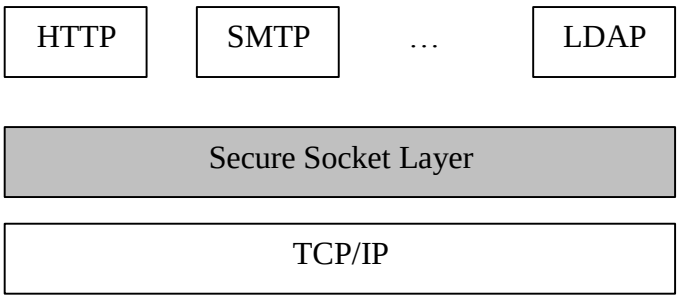
Υπάρχουν περισσότεροι από έναν τρόποι για δύο μέλη να εγκαθιδρύσουν ένα κοινό μυστικό, αλλά ο IKE πάντα χρησιμοποιεί μία ανταλλαγή Diffie-Hellman. Η πράξη του να κάνει μία DH ανταλλαγή δεν είναι διαπραγματεύσιμη, παρά μόνο οι παράμετροι που θα χρησιμοποιηθούν είναι. Το IKE δανείζεται πέντε ομάδες από το Oakley, οι τρεις είναι παραδοσιακές ανταλλαγές που κάνουν εκθετικοποίηση κατά modulo σε πολύ μεγάλο πρώτο αριθμό και οι άλλες δύο είναι ομάδες ελλειπτικών καμπυλών. Η ανταλλαγή Diffie-Hellman και η εγκαθίδρυση ενός κοινού μυστικού είναι το δεύτερο βήμα του πρωτοκόλλου IKE.

Κατά την ολοκλήρωση της ανταλλαγής DH τα δύο μέλη έχουν ένα κοινό μυστικό αλλά δεν είναι αυθεντικοποιημένοι. Μπορούν να το χρησιμοποιήσουν για να προστατέψουν την επικοινωνία τους, αλλά δεν έχουν καμία εγγύηση ότι το απομακρυσμένο μέλος είναι στη πραγματικότητα κάποιος που εμπιστεύεται. Το επόμενο βήμα στην ανταλλαγή IKE είναι η αυθεντικοποίηση του DH κοινού μυστικού και επομένως η αυθεντικοποίηση του ίδιου του συσχετισμού ασφάλειας IKE. Υπάρχουν πέντε μέθοδοι αυθεντικοποίησης που καθορίζονται στον IKE: κλειδιά που έχουν προ διανεμηθεί, η ψηφιακή υπογραφή που χρησιμοποιεί το (Digital Signature Standard - DSS), η ψηφιακή υπογραφή που χρησιμοποιεί τον αλγόριθμο δημοσίου κλειδιού RSA, μία ανταλλαγή κρυπτογραφημένου αριθμού (που χρησιμοποιείται μόνο μία φορά) χρησιμοποιώντας τον RSA και μία αναθεωρημένη μέθοδο αυθεντικοποίησης με κρυπτογραφημένους αριθμούς που είναι ελαφρά διαφορετική από την άλλη μέθοδο των κρυπτογραφημένων αριθμών. (Οι αριθμοί που μόλις αναφέραμε είναι απλά τυχαίοι-random αριθμοί) [21].

4.2.4 Το Πρωτόκολλο SSL

Το πρωτόκολλο TCP/IP είναι το κυρίαρχο για μετάδοση και δρομολόγηση πληροφοριών στα πλαίσια του διαδικτύου. Άλλα πρωτόκολλα, όπως π.χ. το SMTP (διακίνησης αλληλογραφίας), το HTTP (διακίνησης υπερκειμένου), LDAP (υπηρεσίες καταλόγου) κ.λπ. εκτελούνται «πάνω» από το TCP/IP, με την έννοια ότι χρησιμοποιούν το TCP/IP για τη διακίνηση δεδομένων. Βάσει της στρωματοποιημένης αρχιτεκτονικής ISO/OSI για τα δίκτυα υπολογιστών, το πρωτόκολλο TCP/IP καλύπτει τα επίπεδα 3 και 4 (δρομολόγησης και μεταφοράς), ενώ τα SMTP, HTTP, LDAP κλπ. λειτουργούν στο επίπεδο 7 (εφαρμογής). Το πρωτόκολλο SSL παρεμβάλλεται μεταξύ των πρωτοκόλλων εφαρμογής και του TCP/IP (όπως φαίνεται στο σχήμα που ακολουθεί), χρησιμοποιώντας το πρωτόκολλο TCP/IP για λογαριασμό των πρωτοκόλλων υψηλότερου επιπέδου και παρέχοντας την εξής επιπρόσθετη λειτουργικότητα:

- Πιστοποίηση του εξυπηρέτη προς τον εξυπηρετούμενο.
- Πιστοποίηση του εξυπηρετούμενου προς τον εξυπηρέτη.
- Κρυπτογράφηση της επικοινωνίας.



Εικόνα 26. Η θέση του SSL στο ISO/OSI[29].

Η λειτουργικότητα αυτή είναι θεμελιώδης για ασφαλή επικοινωνία στο διαδίκτυο για τους κάτωθι λόγους:

1. *Πιστοποίηση του εξυπηρέτη προς τον εξυπηρετούμενο*: Ο εξυπηρετούμενος μπορεί να διακριβώσει την ταυτότητα του εξυπηρέτη. Το λογισμικό του εξυπηρετούμενου μπορεί να χρησιμοποιήσει ένα σύνολο από τεχνικές κρυπτογραφίας δημόσιου κλειδιού για να ελέγξει ότι το πιστοποιητικό και η δημόσια ταυτότητα του εξυπηρέτη είναι έγκυρα και έχουν εκδοθεί από μία αρχή πιστοποίησης την οποία ο εξυπηρετούμενος εμπιστεύεται. Η σημασία του ελέγχου μπορεί να είναι μεγάλη, αν π.χ. αποστέλλονται αριθμοί πιστωτικών καρτών ή απόρρητα δεδομένα και πρέπει να εξασφαλισθεί ότι μόνο ο προτιθέμενος εξυπηρέτης τα λαμβάνει.
2. *Πιστοποίηση του εξυπηρετούμενου προς τον εξυπηρέτη*: Ο εξυπηρέτης διακριβώνει την ταυτότητα του χρήστη, με τις ίδιες τεχνικές που χρησιμοποιούνται για την πιστοποίηση του εξυπηρέτη προς τον εξυπηρετούμενο. Έτσι ελέγχεται ότι το πιστοποιητικό και η δημόσια ταυτότητα του εξυπηρετούμενου είναι έγκυρα και έχουν εκδοθεί από μία αρχή πιστοποίησης την οποία ο εξυπηρέτης εμπιστεύεται. Η σημασία του ελέγχου μπορεί να είναι μεγάλη, αν π.χ. αποστέλλονται εμπιστευτικά δεδομένα και ο εξυπηρέτης θέλει να εξασφαλίσει ότι μόνο ο προτιθέμενος παραλήπτης τα λαμβάνει.
3. *Κρυπτογράφηση της επικοινωνίας*: Το πρωτόκολλο SSL κρυπτογραφεί όλη την επικοινωνία μεταξύ εξυπηρέτη και εξυπηρετούμενου. Τα δεδομένα κρυπτογραφούνται από τον αποστολέα και αποκρυπτογραφούνται από τον παραλήπτη, επιτυγχάνοντας έτσι υψηλό βαθμό εμπιστευτικότητας. Η εμπιστευτικότητα είναι σημαντική και για τους δύο ενεχόμενους σε οποιαδήποτε ιδιωτική συναλλαγή. Επιπρόσθετα, όλα τα δεδομένα που αποστέλλονται μέσω μιας κρυπτογραφημένης σύνδεσης SSL προστατεύονται με ένα μηχανισμό για ανίχνευση παρεμβάσεων, εντοπισμό δηλαδή προσπαθειών για αλλοίωσή τους κατά τη μεταφορά.

Το πρωτόκολλο SSL περιλαμβάνει δύο επί μέρους πρωτόκολλα: το *πρωτόκολλο εγγραφών SSL* και το *πρωτόκολλο χειραψίας SSL*. Το πρωτόκολλο εγγραφών SSL καθορίζει τη μορφή που χρησιμοποιείται για τη μετάδοση των δεδομένων. Το πρωτόκολλο χειραψίας SSL ορίζει μία ακολουθία μηνυμάτων που πρέπει να ανταλλαχθούν μεταξύ εξυπηρέτη και εξυπηρετούμενου προκειμένου να εγκαθιδρυθεί μία σύνδεση μεταξύ τους. Τα μηνύματα αυτά ανταλλάσσονται με στόχο:

- Να πιστοποιηθεί ο εξυπηρέτης στον εξυπηρετούμενο.
- Να επιτραπεί στον εξυπηρέτη και στον εξυπηρετούμενο να συμφωνήσουν πάνω στους αλγόριθμους κρυπτογραφίας που θα χρησιμοποιηθούν για την επικοινωνία.
- Προαιρετικά, να πιστοποιηθεί ο εξυπηρετούμενος στον εξυπηρέτη.
- Να δημιουργηθούν «διαμοιραζόμενα μυστικά» μέσω τεχνικών κρυπτογραφίας δημόσιου κλειδιού. Τα «διαμοιραζόμενα μυστικά» θα χρησιμοποιηθούν για την κρυπτογράφηση της επικοινωνίας.
- Εγκαθίδρυση του κρυπτογραφημένου διαύλου επικοινωνίας.

4.2.5 Η Χειραψία του Πρωτοκόλλου SSL

Το πρωτόκολλο SSL χρησιμοποιεί έναν συνδυασμό κρυπτογραφίας δημοσίου κλειδιού και συμμετρικής κρυπτογραφίας. Η συμμετρική κρυπτογραφία είναι ταχύτερη, αλλά η κρυπτογραφία

δημοσίου κλειδιού παρέχει καλύτερες τεχνικές διακρίβωσης ταυτότητας. Μία σύνοδος SSL ξεκινά πάντα με ανταλλαγή μηνυμάτων που ονομάζεται *χειραψία SSL*. Η χειραψία επιτρέπει στον εξυπηρετούμενο να πιστοποιήσει την ταυτότητα του εξυπηρέτη και στη συνέχεια υποστηρίζει τη συνεργασία μεταξύ εξυπηρέτη και εξυπηρετούμενου για δημιουργία συμμετρικών κλειδιών που θα χρησιμοποιηθούν για την κρυπτογράφηση, αποκρυπτογράφηση και ανίχνευση παρεμβάσεων στην κρυπτογραφημένη επικοινωνία. Η χειραψία περιλαμβάνει επίσης προαιρετικά την πιστοποίηση της ταυτότητας του εξυπηρετούμενου από τον εξυπηρέτη.

Τα βήματα που περιλαμβάνονται κατά τη διάρκεια της χειραψίας είναι γενικώς τα ακόλουθα (είναι δυνατόν να διαφοροποιούνται λίγο ανάλογα με τις τεχνικές ανταλλαγής κλειδιών):

1. *Client Hello*: Ο εξυπηρετούμενος αποστέλλει στον εξυπηρέτη τον αριθμό έκδοσης του SSL του εξυπηρετούμενου, μία λίστα υποστηριζόμενων αλγόριθμων κρυπτογράφησης και αντιστοίχων μεγεθών κλειδιών, την ταυτότητα της συνόδου κ.τ.λ.
2. *Server Hello*: Ο εξυπηρέτης επιλέγει τον πιο κατάλληλο αλγόριθμο κρυπτογράφησης που υποστηρίζει τόσο αυτός όσο και ο εξυπηρετούμενος και αποστέλλει την ταυτότητα του στον εξυπηρετούμενο. Η αποστελλόμενη ταυτότητα εμπεριέχει και το μήκος των σχετικών κλειδιών.
3. *Certificate*: (προαιρετικό) Αν ο αλγόριθμος κρυπτογράφησης που θα επιλεχθεί απαιτεί πιστοποίηση του εξυπηρέτη, ο εξυπηρέτης αποστέλλει το πιστοποιητικό του στον εξυπηρετούμενο. Το πιστοποιητικό περιέχει το δημόσιο κλειδί του εξυπηρέτη. Βάσει του πιστοποιητικού ο εξυπηρετούμενος μπορεί να διακριβώσει την ταυτότητα του εξυπηρέτη. Αν η διακρίβωση αποτύχει (το πιστοποιητικό είναι άκυρο, έχει λήξει, δεν αντιστοιχεί στον εξυπηρέτη που το έστειλε ή δεν μπορεί να επαληθευθεί από έμπιστη αρχή πιστοποίησης), το σφάλμα σημειώνεται και ο εξυπηρετούμενος πρέπει να αποφασίσει αν θα συνεχίσει ή όχι σε μία κρυπτογραφημένη μεν, μη πιστοποιημένη δε επικοινωνία. Η απόφαση συνήθως λαμβάνεται με προτροπή του χρήστη.
4. *Certificate request*: (προαιρετικό) Αν ο εξυπηρετούμενος ζητά έναν πόρο στον εξυπηρέτη που απαιτεί πιστοποίηση εξυπηρετούμενου, ο εξυπηρέτης αποστέλλει ένα μήνυμα με το οποίο ζητά το πιστοποιητικό του εξυπηρετούμενου.
5. *Server key exchange*: (προαιρετικό) Το μήνυμα αυτό αποστέλλεται μόνο αν το πιστοποιητικό του εξυπηρέτη (που περιέχει το δημόσιο κλειδί του εξυπηρέτη) δεν είναι επαρκές για την ανταλλαγή κλειδιών που θα ακολουθήσει.
6. *Server Hello Done*: Με το μήνυμα αυτό ο εξυπηρέτης υποδεικνύει ότι έχει τελειώσει την προκαταρκτική φάση εγκαθίδρυσης της συνόδου.
7. *Certificate*: (προαιρετικό) Αν ο εξυπηρετούμενος έχει λάβει μήνυμα *Certificate request*, με το μήνυμα αυτό αποστέλλεται το πιστοποιητικό του εξυπηρετούμενου προς τον εξυπηρέτη. Ο εξυπηρέτης θα προβεί στη διακρίβωση της ταυτότητας του εξυπηρετούμενου κατά τρόπο αντίστοιχο με αυτόν που χρησιμοποιεί ο εξυπηρετούμενος για διακρίβωση της ταυτότητας του εξυπηρέτη.
8. *Client key exchange*: Βάσει των μέχρι τώρα ανταλλαγέντων δεδομένων, ο εξυπηρετούμενος δημιουργεί το *προκαταρκτικό μυστικό* (premaster secret) για τη συγκεκριμένη σύνοδο, το κρυπτογραφεί με το δημόσιο κλειδί του εξυπηρέτη και το αποστέλλει σ' αυτόν.
9. *Certificate verify*: (προαιρετικό) Αν ο εξυπηρέτης έχει ζητήσει το πιστοποιητικό του εξυπηρετούμενου, τι μήνυμα αυτό του επιτρέπει να ολοκληρώσει τη διαδικασία επαλήθευσης του πιστοποιητικού.
10. *Change cipher spec*: Ο εξυπηρετούμενος πληροφορεί τον εξυπηρέτη ότι είναι έτοιμος να μεταβεί σε ασφαλή επικοινωνία.
11. *Finished*: Ο εξυπηρετούμενος πληροφορεί τον εξυπηρέτη ότι έχει τελειώσει το δικό του τμήμα της χειραψίας.
12. *Change cipher spec*: Ο εξυπηρέτης πληροφορεί τον εξυπηρετούμενο ότι είναι έτοιμος να μεταβεί σε ασφαλή επικοινωνία.
13. *Finished*: Ο εξυπηρέτης πληροφορεί τον εξυπηρετούμενο ότι έχει τελειώσει το δικό του τμήμα της χειραψίας.

4.2.6 Η Ανταλλαγή Δεδομένων στο Πρωτόκολλο SSL

Μετά το πέρας της χειραψίας ο εξυπηρέτης και ο εξυπηρετούμενος επικοινωνούν μόνο με κρυπτογραφημένα δεδομένα. Η κρυπτογράφηση γίνεται με *συμμετρικό αλγόριθμο*, προκειμένου να επιτυγχάνεται μεγαλύτερη ταχύτητα στην επικοινωνία. Ο συγκεκριμένος συμμετρικός αλγόριθμος που θα χρησιμοποιηθεί έχει αποφασισθεί στο βήμα 2 της χειραψίας (Server hello), όταν ο εξυπηρέτης έχει επιλέξει τον πιο κατάλληλο αλγόριθμο και έχει ενημερώσει τον εξυπηρετούμενο σχετικά.

Για τη δημιουργία των κατάλληλων κλειδιών για τον συμμετρικό αλγόριθμο, αξιοποιείται το προκαταρκτικό μυστικό που αναφέρθηκε στο βήμα 8 (client key exchange) της χειραψίας. Ανακαλέστε ότι ο εξυπηρετούμενος στο βήμα αυτό έχει δημιουργήσει κάποια δεδομένα, τα έχει κρυπτογραφήσει με το δημόσιο κλειδί του εξυπηρέτη (το οποίο έχει εξάγει από το επαληθευμένο πιστοποιητικό του εξυπηρέτη) και τα έχει αποστείλει στον εξυπηρέτη. Ο εξυπηρέτης, ως μόνος κάτοχος του αντίστοιχου ιδιωτικού κλειδιού, είναι ο μόνος που μπορεί να αποκρυπτογραφήσει το μήνυμα αυτό και να εξάγει το προκαταρκτικό μυστικό που απέστειλε ο εξυπηρετούμενος. Έτσι, στο σημείο αυτό εξυπηρέτης και εξυπηρετούμενος κατέχουν από κοινού ένα διαμοιραζόμενο μυστικό, το οποίο δεν είναι γνωστό σε κανέναν άλλο. Στη συνέχεια, εξυπηρέτης και εξυπηρετούμενος προβαίνουν παράλληλα σε μετασχηματισμούς πάνω στο διαμοιραζόμενο μυστικό, έτσι ώστε να καταλήξουν στο τελικό κλειδί συνόδου, το κλειδί δηλαδή για τον συμμετρικό αλγόριθμο κρυπτογράφησης που θα χρησιμοποιηθεί για τη συγκεκριμένη σύνοδο.

Το κλειδί συνόδου είναι το αποτέλεσμα μιας συνάρτησης SessionKey = genKey(premasterSecret, cipher, keyLen) όπου premasterSecret είναι το προκαταρκτικό μυστικό, cipher είναι ο αλγόριθμος κρυπτογράφησης και keyLen το μήκος των κλειδιών που θα χρησιμοποιηθούν στον αλγόριθμο. Με δεδομένο ότι τόσο ο εξυπηρέτης όσο και ο εξυπηρετούμενος χρησιμοποιούν τις ίδιες παραμέτρους στη συνάρτηση αυτή (το προκαταρκτικό μυστικό έχει ανταλλαχθεί στο βήμα 8 της χειραψίας ενώ ο αλγόριθμος κρυπτογράφησης και το μήκος του κλειδιού έχουν ανταλλαχθεί στο βήμα 2 της χειραψίας), το αποτέλεσμα που λαμβάνουν είναι το ίδιο, συνεπώς μπορούν να το χρησιμοποιήσουν ως κλειδί σε συμμετρικό αλγόριθμο κρυπτογραφίας[31,32,33].

5. Πρακτική μελέτη πρωτοκόλλων ανταλλαγής κλειδιού

Στα πλαίσια της διατριβής αυτής και με σκοπό τη πρακτική μελέτη κάποιων από των πιο διαδεδομένων πρωτοκόλλων ανταλλαγής κλειδιών που παρουσιάσαμε και αναλύσαμε στα προηγούμενα κεφάλαια, αναπτύξαμε μια εφαρμογή σε Java SE. Χρησιμοποιήσαμε κυρίως το framework Java Cryptography Architecture (JCA), JCE provider και κάποια στοιχεία από τη συλλογή Bouncy Castle [http://bouncycastle.org/].

Τα πρωτόκολλα που καταφέραμε να υλοποιήσουμε επιτυχώς και να τα δοκιμάσουμε είναι των ασύμμετρων αλγορίθμων RSA, DH και του συμμετρικού DES σε συνδυασμό με τους δύο προηγούμενους.

Στη συνέχεια του κεφαλαίου θα παρουσιάσουμε τις διάφορες δοκιμές που πραγματοποιήσαμε ώστε να δημιουργήσουμε ένα πίνακα με τα συγκεντρωτικά αποτελέσματα των παραπάνω αλγορίθμων όσων αφορά τα επίπεδα απόδοσης τους (μνήμη-ταχύτητα). Η ανάλυση της ασφάλειας των αλγορίθμων δεν εξετάζεται στη παρούσα ενότητα. Έχει παρουσιαστεί αναλυτικά σε θεωρητικό-μαθηματικό επίπεδο σε προηγούμενα κεφάλαια (2 και 3) η ασφάλεια που μπορεί να προσφέρει ο κάθε αλγόριθμος και πως αυτή εξαρτάται από το μέγεθος του κλειδιού τους [24].

5.1 Περιβάλλον Εργασίας

Για την υλοποίηση των πρωτοκόλλων χρησιμοποίηθηκε ένας φορήτος υπολογιστής με λειτούργικο windows 7 32bit.Στην συνέχεια εγκαταστήσαμε το Java developer kit από την επίσηση ιστοσελίδα της Oracle .Το JDK όμως είναι το εργαλείο για να φτιάξουμε προγράμματα γραμμένα σε java. Το JDK περιέχει τον **javac** που είναι ο μεταγλωττιστής που χρειαζόμαστε.Στην επίσημη σελίδα της Oracle μας δίνεται η δυνατότητα μαζί με το java Developer Kiit να κατεβάσουμε και το Netbeans το οποίο είναι ένα ολοκληρωμένο ανάπτυξης εφαρμογών σε java.

Στην συνέχεια από την επίσημη ιστοσελίδα της Oracle θα κατεβάσουμε και θα εγκαταστήσουμε το Java Cryptography Extention.Το οποίο είναι μια επίσημη έκδοση της Oracle, η οποία μας παρέχει ένα πλαίσιο για την υλοποίση διάφορων κρυπτογραγικών αλγόριθμων.Για την εγκατάσταση του Java Cryptography Extention απαιτείται να κατεβάσουμε το αρχείο, να το κάνούμε unzip και από το συγκεκριμένο φάκελο να επιλέξουμε δύο αρχεία και να τα τοποθετήσουμε στο security ο οποιος βρίσκεται Computer\Local disk(C)\Program File\java\jre7\lib\security.Τέλος χρησιμοποιήσαμε κάποια έτοιμα στοιχεία από την συλλογή bouncycastle.

5.2 Τρόπος λειτουργίας εφαρμογής client/server

Στην ενότητα αυτή θα παρουσιάσουμε μερικά σενάρια εκτέλεσης της εφαρμογής ClientServerCryptography. Πριν προχωρήσουμε στα σενάρια θα θέλαμε να παρουσιάσουμε τον τρόπο χρήσης της εφαρμογής.

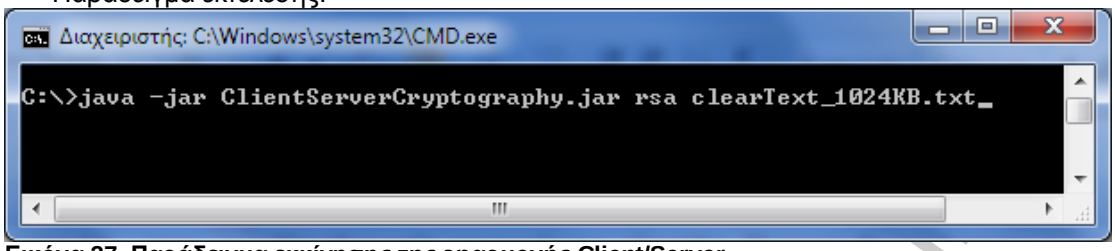
Για να εκκινήσει η εφαρμογή ClientServerCryptography.jar θα πρέπει μέσα από το περιβάλλον του cmd.exe να πάμε στο φάκελο όπου υπάρχει η εφαρμογή και να γράψουμε την εντολή: *java –jar ClientServerCryptography.jar algorithm filename.* Οι παράμετροι *algorithm* και filename είναι υποχρεωτικοί και στη θέση του πρώτου έχουμε τρεις δυνατές επιλογές:

- des (Ανταλλαγή κλειδιών des μέσω του πρωτοκόλλου rsa. Η κρυπτογράφηση του μηνύματος γίνεται με τον συμμετρικό αλγόριθμο des.)
- rsa (Η κρυπτογράφηση του μηνύματος γίνεται με τον αλγόριθμο δημοσίου κλειδιού rsa. Χρησιμοποιείται το δημόσιο και το ιδιωτικό κλειδί του αλγορίθμου για την κρυπτογράφηση/αποκρυπτογράφηση του μηνύματος αντίστοιχα.)
- dh (Ανταλλαγή κλειδιών des μέσω του πρωτοκόλλου dh. Η κρυπτογράφηση του μηνύματος γίνεται με τον συμμετρικό αλγόριθμο des.)

Στη θέση του *filename* θα πρέπει να είναι το όνομα του αρχείου στο οποίο υπάρχει μήνυμα (**με λατινικούς χαρακτήρες**) που θέλουμε να σταλεί από τον client στον server. Εδώ θα θέλαμε να διευκρινίσουμε ότι για την ευκολία του χρήστη, δίνουμε τη δυνατότητα να φορτώνει

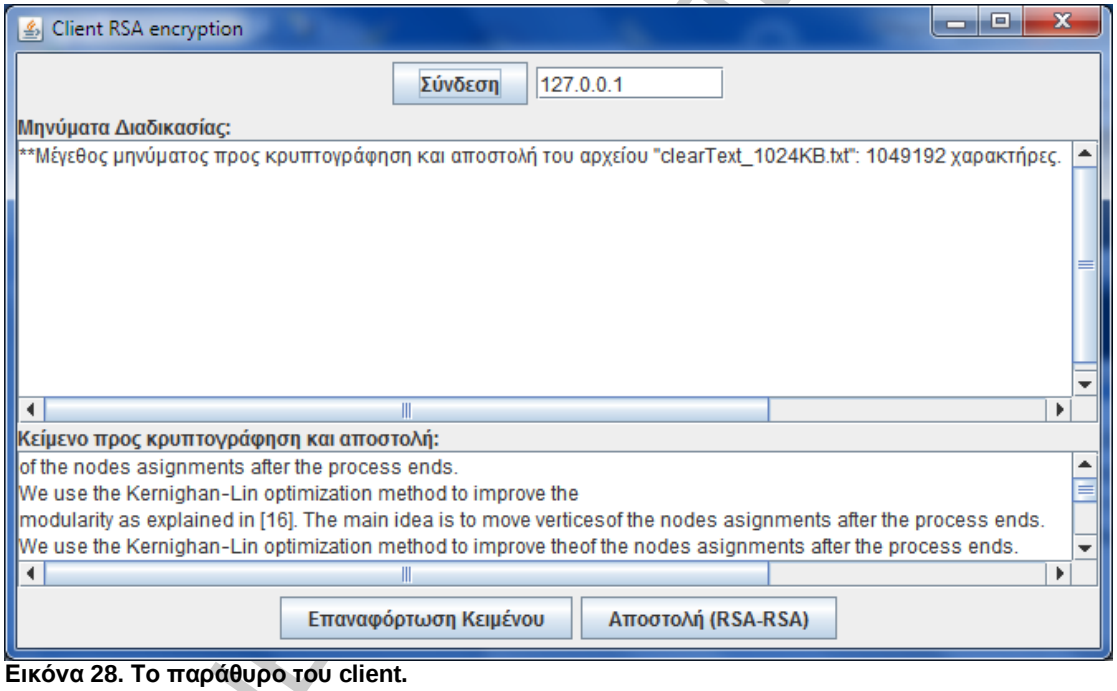
στην εφαρμογή μεγάλο όγκο δεδομένων ώστε να πραγματοποιεί με ευκολία σενάρια δοκιμών των τριών παραπάνω αλγορίθμων. Σημειώνεται ότι ο χρήστης έχει τη δυνατότητα, αφού εκκινήσει η εφαρμογή να γράφει επιπλέον μηνύματα στην περιοχή της επεξεργασίας μηνυμάτων.

Παράδειγμα εκτέλεσης:



Εικόνα 27. Παράδειγμα εκκίνησης της εφαρμογής Client/Server.

Με την παραπάνω εντολή θα εκκινήσει ο client με αρχικοποιημένο το πεδίο του μηνύματος προς αποστολή με το περιεχόμενο του αρχείου clearText_1024KB.txt. Ο αλγόριθμος που θα χρησιμοποιηθεί για την ασύμμετρη κρυπτογράφηση είναι αυτός του rsa. Μετά την εκκίνηση του client εκκινεί και ο server. Παρακάτω θα εξηγήσουμε λίγο τα παράθυρα του client και του server. Εκκινεί αρχικά ο client και στη συνέχεια ο server που θα κάνει την αποκρυπτογράφηση. Να σημειώσουμε ότι ο client κρυπτογραφεί το μήνυμα με τον αλγόριθμο που έχουμε επιλέξει κατά την εκκίνηση και ο server παραλαμβάνει το μήνυμα και το αποκρυπτογραφεί.

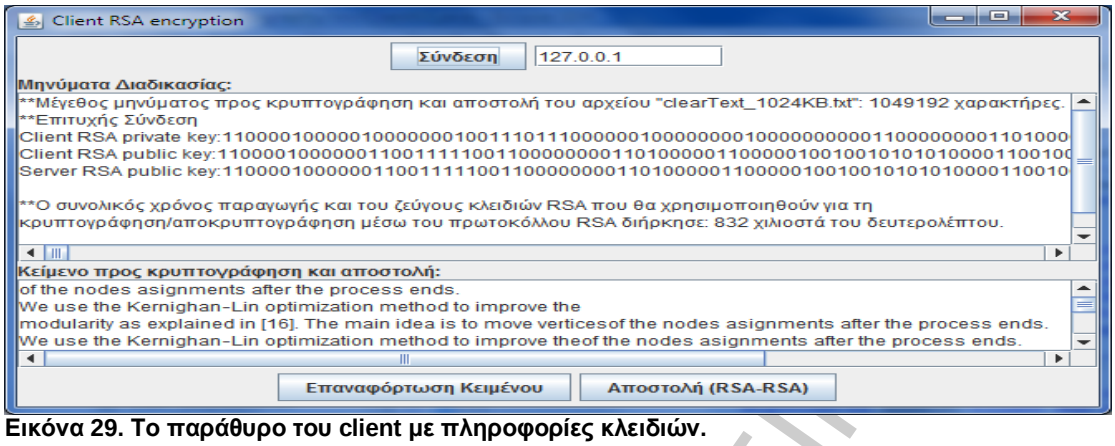


Εικόνα 28. Το παράθυρο του client.

Στη παραπάνω Εικόνα βλέπουμε το παράθυρο του client με το τίτλο που υποδεικνύει το ρόλο και τον αλγόριθμο που θα χρησιμοποιεί για την αποστολή μηνυμάτων. Στην αρχή του παραθύρου βλέπουμε το κουμπί «Σύνδεση» με το οποίο αφού πληκτρολογήσουμε την ip του server το πατάμε. Επειδή ο server τρέχει στο ίδιο μηχάνημα τοπικά, συμπληρώνουμε τη διεύθυνση localhost.

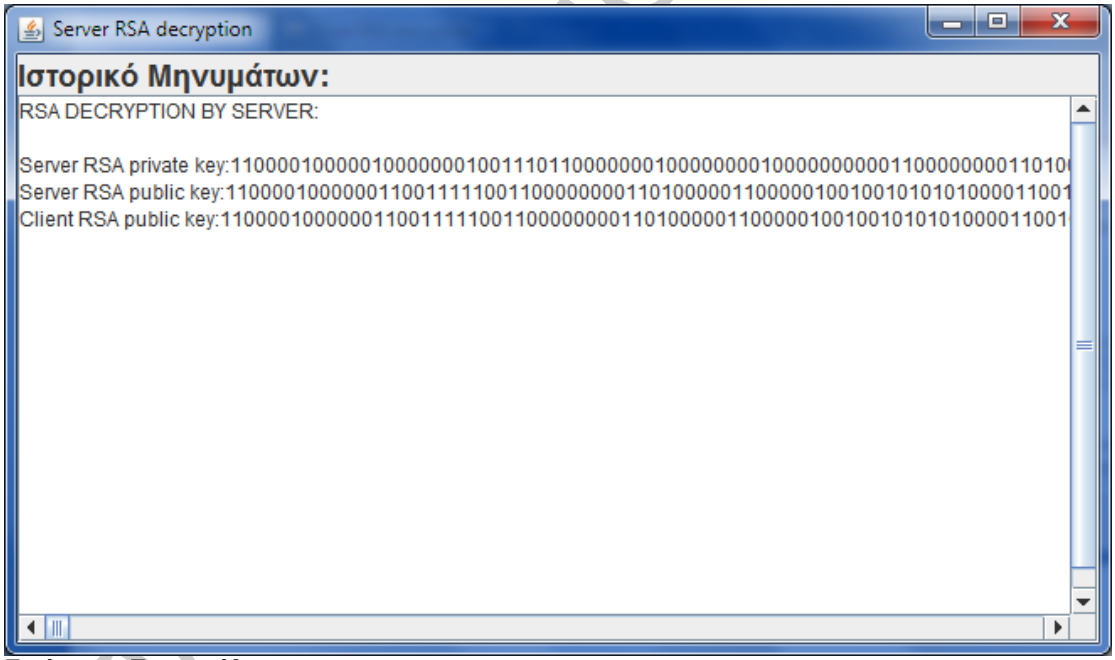
Στη συνέχεια το παράθυρο χωρίζεται σε δύο μέρη. Στο πρώτο εμφανίζονται μηνύματα που αφορούν την εξέλιξη της διαδικασίας καθώς και σημαντικές πληροφορίες αποδόσεων των αλγορίθμων. Στο δεύτερο είναι το μέρος που συμπληρώνουμε το μήνυμα που θέλουμε να στείλουμε στο server. Όταν ξεκινάει ο client φορτώνει το περιεχόμενο του αρχείου που βάλαμε σαν δεύτερη παράμετρο κατά την εκκίνηση της εφαρμογής. Μπορούμε να το επεξεργαστούμε πριν το στείλουμε ή να το διαγράψουμε εντελώς και να γράψουμε κάτι άλλο. Με το κουμπί «Επαναφόρτωση» έχουμε τη δυνατότητα να τραβήξουμε όσες φορές θέλουμε το περιεχόμενο του αρχείου και με αυτό τον τρόπο να πολλαπλασιάζουμε σταθερά το μέγεθός του.

Το πρώτο μήνυμα που εμφανίζεται στα «Μηνύματα Διαδικασίας» αφορά το όνομα του αρχείου του οποίου φορτώσαμε τα δεδομένα, καθώς και το πλήθος των χαρακτήρων του. Αφού πατήσουμε το κουμπί «Σύνδεση», στα Μηνύματα Διαδικασίας θα δούμε τις παρακάτω πληροφορίες:



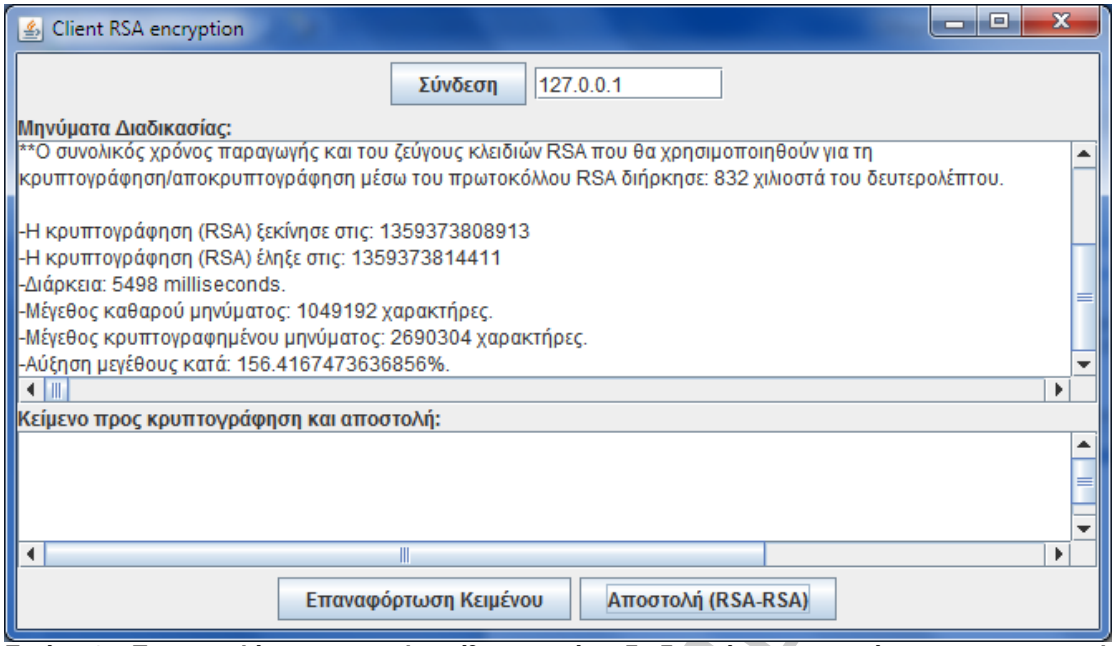
Εικόνα 29. Το παράθυρο του client με πληροφορίες κλειδιών.

που αφορούν τα κλειδιά του αλγορίθμου των δύο πλευρών. Επίσης η τελευταία πληροφορία που εμφανίζεται στο πλαίσιο αφορά το συνολικό χρόνο που διήρκησε η παραγωγή και η ανταλλαγή των κλειδιών ώστε να είναι σε θέση να ξεκινήσει η κρυπτογράφηση του μηνύματος και η αποστολή του με ασφάλεια. Ας δούμε τώρα την Εικόνα του server, στον οποίο εμφανίζονται και σε αυτόν οι πληροφορίες των κλειδιών του αλγορίθμου όταν ο χρήστης πιέσει το κουμπί «Σύνδεση».



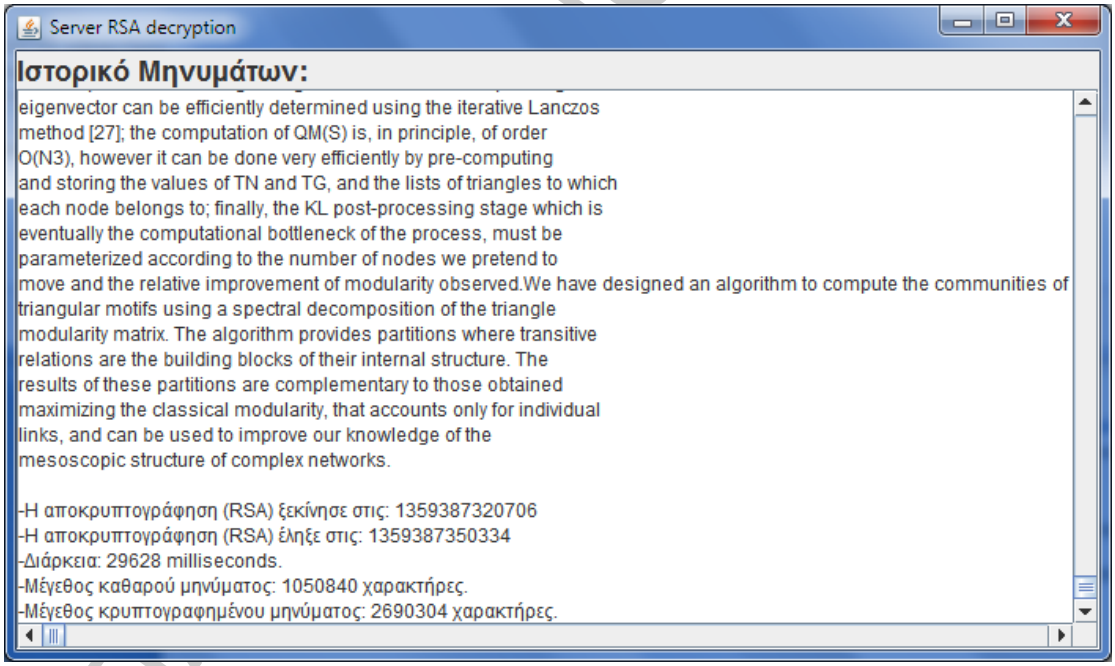
Εικόνα 30. Το παράθυρο του server.

Τέλος ας εξετάσουμε τι γίνεται στον client και στο server όταν πιέσουμε το «Αποστολή (...)»:



Εικόνα 31. Τα αποτελέσματα του αλγορίθμου κατά τη διαδικασία κρυπτογράφησης και αποστολής του μηνύματος.

Στην Εικόνα 31. παρατηρούμε ότι με την «Αποστολή» παρουσιάζονται στον client τα αποτελέσματα της κρυπτογράφησης και στην Εικόνα 32. βλέπουμε τα αποτελέσματα της διαδικασίας αποκρυπτογράφησης. Δηλαδή το καθαρό κείμενο και τα αποτελέσματα (χρόνου-μνήμης) της διαδικασίας αποκρυπτογράφησης.



Εικόνα 32. Τα αποτελέσματα της διαδικασίας αποκρυπτογράφησης στον server.

5.3 Δοκιμές

Στην ενότητα αυτή θα επιχειρήσουμε να διεξάγουμε ένα σενάριο εκτέλεσης για κάθε ένα πρωτόκολλο που υλοποιήσαμε στην εφαρμογή που παρουσιάσαμε πριν. Εκτός από τα τελικά αποτελέσματα που θα εξάγουμε, θα αναλύσουμε τα βήματα της εφαρμογής εντοπίζοντας «κρίσιμα» σημεία της διαδικασίας ανταλλαγής και εγκαθίδρυσης των κρυπτογραφικών κλειδιών.

Το πρώτο μας σενάριο αφορά τον αλγόριθμο RSA. Η κρυπτογράφηση που θα πραγματοποιήσουμε θα γίνει με το δημόσιο κλειδί του RSA και όχι με κάποιο κλειδί DES που θα συμφωνηθεί μέσω του RSA. Η κλάση *KeyPairGenerator* επιτρέπει τη χρήση κλειδιού για τον RSA με μήκη 1024 ή 2048 bits. Στην υλοποίηση μας παράγουμε κλειδιά των 1024 bits στη κλάση RSA:

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(1024); //keysize
KeyPair kp = kpg.genKeyPair();
pbkey = kp.getPublic();
prkey = kp.getPrivate();
```

Εικόνα 33. Παραγωγή κλειδιών RSA σε JCA.

Το πρώτο βήμα είναι να επιλέξουμε το μέγεθος του αρχείου/κειμένου το οποίο θα κρυπτογραφήσουμε, θα αποστείλουμε στο server και αυτός με τη σειρά του θα το αποκρυπτογραφήσει. Προετοιμάσαμε ένα κείμενο μεγέθους 2048KB = 2MB και θα το βάλουμε σαν παράμετρο στην εφαρμογή μας. Άρα θα γράψουμε την εντολή: *java –jar ClientServerCryptography.jar rsa clearText_2048KB.txt*

Στο κώδικα της εφαρμογής εκτυπώνουμε κάθε στιγμή το στάδιο στο οποίο βρίσκεται η εξέλιξη της διαδικασίας ώστε να παρουσιάσουμε την εικόνα των βημάτων ανάλογα τον αλγόριθμο που επιλέγεται. Στη Εικόνα 34. φαίνονται τα βήματα με τα αντίστοιχα τμήματα κώδικα της εφαρμογής όπου αυτό κρίνεται σκόπιμο.

Βήμα 1. Δημιουργία Client UI.

Βήμα 2. Δημιουργία στιγμιοτύπου RSA στον client (clientKeyRSA).

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(1024);
KeyPair kp = kpg.genKeyPair();
pbkey = kp.getPublic();
prkey = kp.getPrivate();
cipher = Cipher.getInstance("RSA");
```

Σχόλια: Παράγεται το ζεύγος κλειδιών RSA στον client.

Βήμα 3. Δημιουργία στιγμιοτύπου RSA στον server.

Σχόλια: Ακριβώς το ίδιο που συμβαίνει στο βήμα 2., για τον server όμως αυτή τη φορά.

Βήμα 4. Δημιουργία των κατάλληλων socket επικοινωνίας στη περίπτωση RSA από τη πλευρά του server.

Βήμα 5. Δημιουργία νήματος κλειδιών RSA - RSA στον server.

Βήμα 6. Δημιουργία Socket clientMsg και clientDES.
Βήμα 7. Δημιουργία Socket clientRSA.

Σχόλια: Δημιουργούνται τα κατάλληλα socket επικοινωνίας για τη συμφωνία και την ανταλλαγή των μηνυμάτων.

Βήμα 8. Δημιουργία νήματος κλειδιών RSA - DES στον client.

Σχόλια: Είτε στη περίπτωση που η κρυπτογράφηση του κειμένου γίνεται με το μυστικό κλειδί DES, είτε με το δημόσιο κλειδί του RSA, στην εφαρμοφή χρησιμοποιούμε για χάρη ευκολίας το ίδιο νήμα, χωρίς να επηρεάζεται η ορθή λειτουργία της εφαρμογής.

Βήμα 9. Δημιουργία streams για τους RSA και DES στον client.

Σχόλια: Συμβαίνει για τον ίδιο λόγο με το Βήμα 8.

Βήμα 10. Στο OutputStream του client για τον RSA εγγράφετε το δημόσιο κλειδί του.

```
ObjOSRSA.writeObject(clientKeyRSA.getPublicKey());
```

Βήμα 11. Ο server διαβάζει το δημόσιο κλειδί RSA του client.

```
clientPBK = (RSAPublicKey) ObjISRSA.readObject();
```

Βήμα 12. Ο client διαβάζει το δημόσιο κλειδί RSA του server.

```
serverPBK = (RSAPublicKey) ObjISRSA.readObject();
```

Βήμα 13. Ο server στέλνει το δημόσιο κλειδί RSA.

```
ObjOSRSA.writeObject(serverKeyRSA.getPublicKey());
```

Βήμα 14. Κρυπτογράφηση μηνύματος με το δημόσιο κλειδί του RSA στον client.

```
byte[] eData = clientKeyRSA.encryptRSA(serverPBK, data);
```

Σχόλια: Στο Βήμα 3. είχαμε δημιουργήσει ένα αντικείμενο RSA. Το αντικείμενο αυτό διαθέτει την encryptRSA η οποία είναι μία μέθοδος που κατασκευάσαμε (εκτός JCA) με σκοπό την κρυπτογράφηση με το δημόσιο κλειδί του RSA.

Βήμα 15. Το κρυπτογραφημένο μήνυμα RSA εγγράφεται στο OutputStream του client.

```
ObjOSMsg.writeObject(eData);
```

Βήμα 16. Το κρυπτογραφημένο κείμενο RSA που έστειλε ο client αποκρυπτογραφείται στον server.

```
data = serverKeyRSA.decryptRSA(serverKeyRSA.getPrivateKey(), eData);
```

Εικόνα 34. Βήματα εκτέλεσης κρυπτογραφημένης επικοινωνίας μεταξύ client/server με τον αλγόρι -θμο RSA.

Με την ίδια λογική και το ίδιο αρχείο-κείμενο που χρησιμοποιήσαμε στον RSA, θα προχωρήσουμε στη δοκιμή του RSA – DES, δηλαδή εκείνου του πρωτοκόλλου που η ανταλλαγή του μυστικού κλειδιού DES γίνεται με τον RSA. Δηλαδή το κλειδί του DES κρυπτογραφείτε με τον RSA, ανταλλάσσεται με τον server και γίνεται εφικτή η κρυπτογραφημένη επικοινωνία μεταξύ των μερών. Επομένως, το μήνυμα που ανταλλάσσεται είναι κρυπτογραφημένο με τον συμμετρικό αλγόριθμο DES.

Βήμα 1. Δημιουργία Client UI.

Βήμα 2. Δημιουργία στιγμιοτύπου RSA στον client (clientKeyRSA).

Βήμα 3. Δημιουργία στιγμιοτύπου RSA στον server.

Βήμα 4. Δημιουργία στιγμιοτύπου DES στον server.

Βήμα 5. Στο γεννήτορα της κλάσης DES παράγεται το μυστικό κλειδί μήκους 56 bit.

```
KeyGenerator kg = KeyGenerator.getInstance("DES");
kg.init(56);
sKey = kg.generateKey();
```

Σχόλια: Η KeyGenerator για τον DES υποστηρίζει μέγεθος κλειδιού = 56 bit. Τα υπόλοιπα 8 bit χρησιμοποιούνται για έλεγχο ισοτιμίας.

Βήμα 6. Ο server κατέχει το μυστικό κλειδί DES.

```
keyDES = DES.getSecretKey();
```

Σχόλια: Το κλειδί έχει δημιουργηθεί ήδη από το Βήμα 4.

Βήμα 7. Δημιουργία κατάλληλων socket.
Βήμα 8. Δημιουργία νήματος κλειδιών RSA - DES στον server.
Βήμα 9. Δημιουργία socket clientMsg και clientDES.
Βήμα 10. Δημιουργία socket clientRSA.
Βήμα 11. Δημιουργία νήματος κλειδιών RSA - DES στον client.
Βήμα 12. Δημιουργία streams για τους RSA - DES στον client.

Βήμα 13. Στο OutputStream για τον RSA στον client εγγράφετε το δημόσιο κλειδί του.

```
ObjOSRSA.writeObject(clientKeyRSA.getPublicKey());
```

Βήμα 14. Ο server διαβάζει το δημόσιο κλειδί RSA του client.

```
clientPBK = (RSAPublicKey) ObjISRSA.readObject();
```

Βήμα 15. Δημιουργείται ένα cipher instance για τον RSA μετασχηματισμό στον client.

```
cipher = Cipher.getInstance("RSA");
```

Βήμα 16. Αρχικοποίηση cipher με το μυστικό κλειδί του RSA στον client.

```
cipher.init(Cipher.UNWRAP_MODE, clientKeyRSA.getPrivateKey());
```

Βήμα 17. Ο server στέλνει το δημόσιο κλειδί RSA.

```
ObjOSRSA.writeObject(serverKeyRSA.getPublicKey());
```

Βήμα 18. Ο server κάνει wrap το μυστικό κλειδί DES με το δημόσιο RSA κλειδί του client.

```
keyDESbyte = RSA.wrapkey(keyDES, clientPBK);
```

Βήμα 19. Ο client διαβάζει το δημόσιο κλειδί RSA του server.

```
serverPBK = (RSAPublicKey) ObjISRSA.readObject();
```

Βήμα 20. Ο server γράφει στο κατάλληλο stream τα byte του κρυπτογραφημένου κλειδιού DES.

```
ObjOSDES.writeObject(keyDESbyte);
```

Βήμα 21. Ο client διαβάζει από το InputStream το κλειδί του DES σε μορφή byte.

```
bkeyDES = (byte[]) ObjISDES.readObject();
```

Βήμα 22. Δημιουργείται το μυστικό κλειδί του DES στον client.

```
keyDES = (SecretKey) cipher.unwrap(bkeyDES, "DES",Cipher.SECRET_KEY);
```

Βήμα 23. Κρυπτογράφηση μηνύματος στον client με το μυστικό κλειδί του DES που έχει συμφωνηθεί ασφαλώς.

```
byte[] eData = DES.SEnc(keyDES, "ENC", data);
```

Βήμα 24. Το κρυπτογραφημένο κείμενο DES που έστειλε ο client μέσω της συμφωνίας RSA αποκρυπτογραφείται στον server.

```
data = DES.SEnc(keyDES, "DEC", eData);
```

Εικόνα 35. Βήματα εκτέλεσης κρυπτογραφημένης επικοινωνίας μεταξύ client/server με το πρωτόκολλο RSA-DES.

Το τελευταίο σενάριο περιλαμβάνει το πρωτόκολλο DH συμφωνίας κλειδιού σε συνδυασμό με τον συμμετρικό αλγόριθμο DES. Κι εδώ αφού συμφωνηθούν τα κλειδιά κρυπτογράφησης με τον DH, χρησιμοποιείται ο αλγόριθμος DES για την κρυπτογράφηση/αποκρυπτογράφηση του κειμένου.

Βήμα 1. Δημιουργία Client UI.

Βήμα 2. Δημιουργία στιγμιοτύπου DH στον client (paramsDH).

Βήμα 3. Με τη κλάση DH παράγονται οι παράμετροι του πρωτοκόλλου.

```
paramGen = AlgorithmParameterGenerator.getInstance("DH");
paramGen.init(512);
params = paramGen.generateParameters();
dhSkipParamSpec = (DHParameterSpec)
params.getParameterSpec(DHParameterSpec.class);
```

Σχόλια: Στο γεννήτορα της κλάσης DH δημιούργουνται οι σχετικοί παράμετροι κλειδιού. Το μήκος του επιλέγεται να είναι τα 512 bits.

Βήμα 4. Δημιουργία κατάλληλων socket για τη περίπτωση DH στον server.
Βήμα 5. Δημιουργία νήματος κλειδιών DH - DES στον server.
Βήμα 6. Δημιουργία socket clientMsg και clientDES.
Βήμα 7. Δημιουργία socket clientDH.
Βήμα 8. Δημιουργία νήματος κλειδιών DH - DES στον client.
Βήμα 9. Δημιουργία streams για τους DH και DES στον client.

Βήμα 10. Δημιουργείται στον client το ζεύγος κλειδιών του DH.

```
KeyPairGenerator clientKpairGen = KeyPairGenerator.getInstance("DH");
clientKpairGen.initialize(paramsDH.getDhSkipParamSpec());
KeyPair clientKpair = clientKpairGen.generateKeyPair();
```

Βήμα 11. Ο server έχει λάβει το δημόσιο κλειδί του client σε κωδικοποιημένη μορφή.Με το υλικό του κλειδιού αρχικοποιεί ένα DH δημόσιο κλειδί.

```
byte[] clientPubKeyEncDH = (byte[]) ObjISDH.readObject();
KeyFactory serverKeyFac = KeyFactory.getInstance("DH");
X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec
(clientPubKeyEncDH);
PublicKey clientPubKey = serverKeyFac.generatePublic(x509KeySpec);
```

Βήμα 12. Αρχικοποιείται η συμφωνία κλειδιού στον DH.

```
keyAgreeDH = KeyAgreement.getInstance("DH");
keyAgreeDH.init(clientKpair.getPrivate());
```

Βήμα 13. Ο client κωδικοποιεί το δημόσιο κλειδί του DH και το στέλνει στο server.

```
clientPubKeyEncDH = clientKpair.getPublic().getEncoded();
```

Βήμα 14. Ο client γράφει στο OutputStream το κωδικοποιημένο δημόσιο κλειδί του DH.

```
ObjOSDH.writeObject(clientPubKeyEncDH);
```

Βήμα 15. Ο client διαβάζει από το InputStream το κωδικοποιημένο δημόσιο κλειδί του server για τον DH.

```
byte[] serverPubKeyEncDH = (byte[]) ObjISDH.readObject();
```

Βήμα 16. Ο server λαμβάνει του παραμέτρους DH που συνδέονται με το δημόσιο κλειδί του client.Θα πρέπει να χρησιμοποιήσει τις ίδιες παραμέτρους όταν παράγει το δικό του ζεύγος κλειδιών DH.

```
DHParameterSpec dhParamSpec = ((DHPublicKey) clientPubKey).getParams();
```

Βήμα 17. Ο server παράγει το δικό του ζεύγος κλειδιών DH.

```
KeyPairGenerator serverKpairGen = KeyPairGenerator.getInstance("DH");
serverKpairGen.initialize(dhParamSpec);
KeyPair serverKpair = serverKpairGen.generateKeyPair();
```

Βήμα 18. Ο server δημιουργεί και αρχικοποιεί το δικό του αντικείμενο KeyAgreement.

```
KeyAgreement serverKeyAgree = KeyAgreement.getInstance("DH");
serverKeyAgree.init(serverKpair.getPrivate());
```

Βήμα 19. Ο server κωδικοποιεί το δημόσιο κλειδί του και το στέλνει στον client.

```
byte[] serverPubKeyEncDH = serverKpair.getPublic().getEncoded();
ObjOSDH.writeObject(serverPubKeyEncDH);
```

Βήμα 20. Ο server προχωρά στην επόμενη φάση της συμφωνίας κλειδιού με το δημόσιο κλειδί DH του client.

```
serverKeyAgree.doPhase(clientPubKey, true);
```

Βήμα 21. Ο client προχωρά στην επόμενη φάση της συμφωνίας κλειδιού με το δημόσιο κλειδί DH του server.

```
keyAgreeDH.doPhase(serverPubKeyDH, true);
```

Βήμα 22. Παράγεται στον client το διαμοιραζόμενο μυστικό της συμφωνίας κλειδιού στο DH.

```
byte[] clientSharedSecret = keyAgreeDH.generateSecret();
```

Βήμα 23. Παράγεται στον client το μυστικό κλειδί DES μέσα από τη συμφωνία δημοσίου κλειδιού DH.

```
clientDesKeyDH = keyAgreeDH.generateSecret("DES");
```

Βήμα 24. Παράγεται στον server το διαμοιραζόμενο μυστικό κλειδί της συμφωνίας κλειδιού στο DH.

```
serverKeyAgree.generateSecret(serverSharedSecret, 0);
```

Βήμα 25. Παράγεται στον server το μυστικό κλειδί DES μέσα από τη συμφωνία δημοσίου κλειδιού DH.

```
serverDesKeyDH = serverKeyAgree.generateSecret("DES");
```

Βήμα 26. Κρυπτογράφηση μηνύματος με το μυστικό κλειδί του DES που έχει συμφωνηθεί ασφαλώς μέσω του ασύμμετρου πρωτοκόλλου DH.

```
byte[] eData = DES.SEnc(clientDesKeyDH, "ENC", data);
```

Βήμα 27. Το κρυπτογραφημένο μήνυμα DES εγγράφεται στο OutputStream του client.

```
ObjOSMsg.writeObject(eData);
```

Βήμα 28. Το κρυπτογραφημένο κείμενο DES που έστειλε ο client μέσω της συμφωνίας DH αποκρυπτογραφείται στον server.

```
data = DES.SEnc(serverDesKeyDH, "DEC", eData);
```

Εικόνα 36. Βήματα εκτέλεσης κρυπτογραφημένης επικοινωνίας μεταξύ client/server με το πρωτόκολλο DH-DES.

Οι τρεις παραπάνω δοκιμές είχαν σκοπό να παρουσιάσουν την αλληλουχία των ενεργειών των πρωτοκόλλων που υλοποιεί η εφαρμογή σε πραγματικά σενάρια κρυπτογραφημένης επικοινωνίας μεταξύ δύο μερών. Σε κάθε βήμα της εκτέλεσης σημειώθηκαν «κρίσιμα» τμήματα κώδικα έτσι ώστε να γίνει πιο κατανοητή η διαδικασία, αλλά και ο κώδικας που βρίσκεται στο Παράρτημα Α.

Στη συνέχεια, Εικόνα 37., εκτελέσαμε στο παρασκήνιο επιπλέον πέντε δοκιμές με την εφαρμογή, εναλλάσοντας κάθε φορά το μέγεθος του μηνύματος εισόδου και το είδος της κρυπτογραφημένης επικοινωνίας. Για κάθε σενάριο συγκεντρώσαμε πληροφορίες και από τη πλευρά της κρυπτογράφησης (client), αλλά και από τη πλευρά της αποκρυπτογράφησης (server). Στη Ενότητα 4.1 δείξαμε αναλυτικά πως μπορεί κάποιος να εκτέλεσει ένα σενάριο επικοινωνίας και να διαβάσει τις πληροφορίες απόδοσης από τα δύο παράθυρα της εφαρμογής.

Μέγεθος cipher text / Χρόνος Εκτέλεσης / Χρόνος Δημιουργίας Κλειδιών		**RSA-DES** (Μήκος Κλειδιού 56 bit)	**RSA** (Μήκος Κλειδιού RSA 1024 bit)	**DH-DES** (Μήκος Κλειδιού DH 512 bit)
Σενάριο 1° (μήνυμα 150 bytes)	*Κρυπτογράφηση*	152 bytes / 1 msec / 945 msec	512 bytes / 1 msec / 697 msec	152 bytes / 1 msec / 842 msec
	Αποκρυπτογράφηση	152 byte / 1 msec / 945 msec	512 bytes / 1 msec / 697 msec	152 bytes / 1 msec / 842 msec
Σενάριο 2° (μήνυμα 1024 bytes)	*Κρυπτογράφηση*	1032 bytes / 56 msec / 945 msec	2816 bytes / 3 msec / 697 msec	1032 bytes / 1 msec / 842 msec
	Αποκρυπτογράφηση	1032 bytes / 50 msec / 945 msec	2816 bytes / 32 msec / 697 msec	1032 bytes / 1 msec / 842 msec
Σενάριο 3° (μήνυμα 512 KB – 524596 bytes)	*Κρυπτογράφηση*	524600 bytes / 31 msec / 945 msec	1345280 bytes / 1568 msec / 697 msec	524600 bytes / 31 msec / 842 msec
	Αποκρυπτογράφηση	524600 bytes / 26 msec / 945 msec	1345280 bytes / 13665 msec / 697 msec	524600 bytes / 24 msec / 842 msec
Σενάριο 4° (μήνυμα 1024 KB – 1049192 bytes)	*Κρυπτογράφηση*	1049200 bytes / 62 msec / 945 msec	2690304 bytes / 4682 msec / 697 msec	1049200 bytes / 61 msec / 842 msec
	Αποκρυπτογράφηση	1049200 bytes / 53 msec / 945 msec	2690304 bytes / 29379 msec / 697 msec	1049200 bytes / 51 msec / 842 msec
Σενάριο 5° (μήνυμα 2048 KB – 2098384 bytes)	*Κρυπτογράφηση*	2098392 bytes / 123 msec / 945 msec	5380352 bytes / 18666 msec / 697 msec	2098392 bytes / 120 msec / 842 msec
	Αποκρυπτογράφηση	2098392 bytes / 102 msec / 945 msec	5380352 bytes / 64289 msec / 697 msec	2098392 bytes / 105 msec / 842 msec

Εικόνα 37. Πίνακας αποτελεσμάτων σεναρίων εκτέλεσης των πρωτοκόλλων DES, RSA-DES, DH-DES.

Για να γίνει πιο κατανοητός ο πίνακας της Εικόνας 37. θα ερμηνεύσουμε τα στοιχεία της πρώτης γραμμής του πίνακα (Σενάριο 1°). Με τον ίδιο τρόπο ο αναγνώστης θα μπορεί να ερμηνεύσει και τις υπόλοιπες. Στη πρώτη στήλη, εκτός από την ένδειξη «Σενάριο 1°» φαίνεται μέσα σε παρένθεση και το μέγεθος του καθαρού κειμένου-μηνύματος (150 bytes). Σε αυτό το σενάριο, όπως και σε κάθε ένα από τα υπόλοιπα, αντιστοιχούν δύο γραμμές, η πρώτη που αναφέρει στοιχεία για τη διαδικασία της κρυπτογράφησης και η δεύτερη στοιχεία για τη διαδικασία της αποκρυπτογράφησης. Για κάθε σενάριο (ίδιο μέγεθος εισόδου = καθαρού κειμένου) κάναμε δοκιμές και για τα τρία πρωτόκολλα που υποστηρίζει η εφαρμογή.

Η πρώτη από τις δοκιμές αναφέρεται στο πρωτόκολλο RSA-DES, όπου χρησιμοποιείται ο RSA για τη κρυπτογραφημένη μετάδοση του μυστικού κλειδιού DES και στη συνέχεια με το κλειδί αυτό γίνονται οι διαδικασίες τις κρυπτογράφησης/αποκρυπτογράφησης. Τα στοιχεία έδειξαν ότι το καθαρό κείμενο αυξήθηκε μόνο κατά 2 bytes (κρυπτοκείμενο), ό,τι ο χρόνος δημιουργίας των κλειδιών έλαβε 945 msec και ό,τι ο χρόνος διεκπεραίωσης της διαδικασίας κρυπτογράφησης διήρκησε 1 msec.

Η δεύτερη από τις δοκιμές αναφέρεται μόνο στο πρωτόκολλο RSA. Δηλαδή η κρυπτογράφηση/αποκρυπτογράφηση γίνεται αποκλειστικά με το δημόσιο/ιδιωτικό κλειδί του ίδιου του αλγορίθμου. Εδώ παρατηρούμε ότι έχουμε μία αρκετά μεγάλη αύξηση του αρχικού μηνύματος (362 bytes) επειδή ο RSA λειτουργεί με μικρά τμήματα δεδομένων κάθε στιγμή, ό,τι ο χρόνος διεκπεραίωσης της διαδικασίας παρέμεινε ο ίδιος και ό,τι ο χρόνος παραγωγής των κλειδιών ήταν μικρότερος από πριν. Το τελευταίο είναι αρκετά λογικό αφού δεν έχουμε σε αυτή τη περίπτωση παραγωγή κλειδιών του αλγόριθμου DES.

Στην ίδια λογική κυμαίνεται και η ερμηνεία του πίνακα για το πρωτόκολλο DH-DES. Εδώ υπάρχει συμφωνία κλειδιού DH, με σκοπό να μεταδοθεί με ασφάλεια το μυστικό κλειδί DES για να μπορέσει να γίνει η κρυπτογράφηση/αποκρυπτογράφηση. Παρατηρούμε ότι έχει σχεδόν ίδιες αποδόσεις με αυτές του RSA-DES, μόνο που εδώ δημιουργούνται ταχύτερα τα κλειδιά.

6. Συμπεράσματα

Μία εσφαλμένη αντίληψη που υπάρχει αφορά την εντύπωση ότι η κρυπτογράφηση δημοσίου κλειδιού είναι ασφαλέστερη μέθοδος σχετικά με τη συμμετρική κρυπτογράφηση,ως ανθεκτικότερη σε κρυπταναλυτικές επιθέσεις. Στην πραγματικότητα η ασφάλεια οποιουδήποτε συστήματος κρυπτογράφησης εξαρτάται από το μήκος κλειδιού και, σε κάθε περίπτωση, από την απαιτούμενη υπολογιστική ισχύ από έναν κρυπταναλυτή για να κατορθώσει να κρυπταναλύσει και αποκαλύψει με επιτυχία ένα κρυπτογραφημένο μήνυμα. Η επικράτηση του ασύμμετρου κρυπτοσυστήματος σε βάρος του συμμετρικού αποτελεί μία εσφαλμένη αντίληψη. Και τα δύο συστήματα χρησιμοποιούνται ισόρροπα και κατ' ουδέν τρόπο δεν προβλέπεται η εγκατάλειψη του συμμετρικού συστήματος, ειδικά όταν είναι γνωστή η σημαντική χρονική επιβάρυνση που απαιτείται για την ολοκλήρωση των εκτελούμενων λειτουργιών σε περιβάλλον ασύμμετρου κρυπτοσυστήματος. Επιπλέον, η διανομή κλειδιών είναι εύκολη όταν χρησιμοποιείται κρυπτογραφία δημοσίου κλειδιού σε σύγκριση με τις επιπλέον χειραψίες (handshaking) που απαιτούνται με τα κέντρα διανομής κλειδιών (key distribution centers) για τη συμμετρική κρυπτογράφηση. Στην πραγματικότητα, στα ασύμμετρα συστήματα απαιτείται εκτέλεση κάποιων πρωτοκόλλων, εμπλέκεται κάποιος έμπιστος ενδιάμεσος αντιπρόσωπος και οι διαδικασίες που παρεμβάλλονται δεν είναι απλούστερες ή περισσότερο αποδοτικές από αυτές που απαιτούνται για συμμετρική κρυπτογράφηση.

Το μεγαλύτερο πρόβλημα της συμμετρικής κρυπτογραφίας είναι η συνεννόηση και ανταλλαγή του κλειδιού, χωρίς κάποιος τρίτος να μάθει για αυτό. Η μετάδοση μέσα από το Διαδίκτυο δεν είναι ασφαλής γιατί οποιοσδήποτε γνωρίζει για την συναλλαγή και έχει τακατάλληλα μέσα μπορεί να καταγράψει όλη την επικοινωνία μεταξύ αποστολέα και παραλήπτη και να αποκτήσει το κλειδί. Έπειτα, μπορεί να διαβάσει, να τροποποιήσει και να πλαστογραφήσει όλα τα μηνύματα που ανταλλάσσουν οι δύο ανυποψίαστοι χρήστες. Βέβαια, μπορούν να βασισθούν σε άλλο μέσο επικοινωνίας για την μετάδοση του κλειδιού (π.χ. τηλεφωνία), αλλά ακόμα και έτσι δεν μπορεί να εξασφαλιστεί ότι κανείς δεν παρεμβάλλεται μεταξύ της γραμμής επικοινωνίας των χρηστών. Η ασύμμετρη κρυπτογραφία δίνει λύση σε αυτό το πρόβλημα αφού σε καμία περίπτωση δεν "ταξιδεύουν" στο δίκτυο οι εν λόγω ευαίσθητες πληροφορίες. Επίσης παρατηρείται κλιμάκωση, δηλαδή ο αριθμός των κλειδιών αυξάνεται τετραγωνικά με τον αριθμό των συμμετεχόντων στην ανταλλαγή μυστικών δεδομένων. Άλλο ένα ακόμα πλεονέκτημα των ασύμμετρων κρυπτοσυστημάτων είναι ότι μπορούν να παρέχουν ψηφιακές υπογραφές που δεν μπορούν να αποκηρυχθούν από την πηγή τους.

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

Μειονέκτημα της ασύμμετρης κρυπτογραφίας είναι η ταχύτητα. Κατά κανόνα, η διαδικασίες κρυπτογράφησης και πιστοποίησης ταυτότητας με συμμετρικό κλειδί είναι σημαντικά ταχύτερες από την κρυπτογράφηση και ψηφιακή υπογραφή με ζεύγος ασύμμετρων κλειδιών. Γι' αυτό το λόγο, όπως προτείνεται και από το πρωτόκολλο SSL, χρησιμοποιείται η κρυπτογράφηση δημοσίου κλειδιού για την ανταλλαγή συμμετρικών κλειδιών και στη συνέχεια χρησιμοποιείται η συμμετρική κρυπτογράφηση για την ουσιαστική επικοινωνία. Η ιδιότητα αυτή καλείται διασφάλιση της μη αποκήρυξης της πηγής (nonrepudiation).

Επίσης, τεράστιο μειονέκτημα της ασύμμετρης κρυπτογραφίας είναι η ανάγκη για πιστοποίηση και επαλήθευση των δημόσιων κλείδων από οργανισμούς (Certificate Authority) ώστε να διασφαλίζεται η κατοχή από τους νόμιμους χρήστες (μόνο αυτοί που μπορούν να έχουν πρόσβαση σε δεδομένα). Όταν κάποιος απατεώνας κατορθώσει και ξεγελάσει τον οργανισμό, μπορεί να συνδέσει το όνομα του με την δημόσια κλείδα ενός νόμιμου χρήστη και να προσποιείται την ταυτότητα αυτού του νόμιμου χρήστη. Σε μερικές περιπτώσεις, η ασύμμετρη κρυπτογραφία δεν είναι απαραίτητη και η συμμετρική κρυπτογραφία από μόνη της είναι αρκετή. Τέτοιες περιπτώσεις αφορούν περιβάλλοντα κλειστά, που δεν έχουν σύνδεση με το Διαδίκτυο. Ένας υπολογιστής μπορεί να κρατά τα μυστικά κλειδιά των χρηστών που επιθυμούν να

εξυπηρετηθούν από αυτόν, μια και δεν υπάρχει ο φόβος για κατάληψη της μηχανής από εξωτερικούς παράγοντες. Επίσης, στις περιπτώσεις που οι χρήστες μπορούν να συναντηθούν και να ανταλλάξουν τα κλειδιά ή όταν η κρυπτογράφηση χρησιμοποιείται για τοπική αποθήκευση κάποιων αρχείων, η ασύμμετρη κρυπτογραφία δεν είναι απαραίτητη.

Υστέρα από τη μελέτη που πραγματοποιήσαμε στα διάφορα πρωτόκολλα ανταλλαγής και εγκαθίδρυσης κρυπτογραφικών κλειδιών καταλήξαμε σε κάποια χρήσιμα συμπεράσματα. Κάποια από αυτά τα εξάγαγαμε ύστερα από την εκτέλεση των σεναρίων της Εικόνας 37.

Τα συμπεράσματα που προέκυψαν από τη μελέτη της βιβλιογραφίας μας είναι:

- Ο RSA αποτελεί έναν από τους πιο διαδεδομένους ασύμμετρους αλγόριθμους για τη δημιουργία ψηφιακών υπογραφών και την επαλήθευσή τους καθώς και για την ασφαλή μεταφορά κλειδιών.
- Η εύρεση μίας κατάλληλης ελλειπτικής καμπύλης είναι μία σχετικά δύσκολη διαδικασία. Οι καμπύλες πρέπει να έχουν ορισμένες ιδιότητες για να μπορούν να είναι ασφαλής.
- Ο ECDH έχει πλεονέκτημα ταχύτητας σε σχέση με τον απλό DH, καθώς εξαλείφει την ανάγκη για εκθετικοποίηση κατά modulo. Επίσης το μέγεθος του κλειδιού που απαιτείται για τον ECDH είναι πολύ μικρότερο από αυτό του απλού DH, για το ίδιο επίπεδο ασφάλειας. Από την άλλη πλευρά, η υποκείμενη θεωρία για την αντίληψη και χρήστη των ελλειπτικών καμπύλων είναι πιο δύσκολη από αυτή των διακριτών λογαρίθμων.
- Οι δυναμικές επιθέσεις άγνωστου μοιράσματος κλειδιού μπορούν να εξαλειφθούν με τη χρήση πιστοποιητικών σε συνδυασμό με κάποια αρχή πιστοποίησης.
- Το πρωτόκολλο MTI/C0 αν και φαίνεται ασφαλές, δεν προσφέρει έμμεση πιστοποίηση κλειδιού.
- Τα πρωτόκολλα της οικογένειας AK, σαν το MQV, δεν παρέχουν ρητή επιβεβαίωση κλειδιού και όταν χρησιμοποιούνται στη πράξη θα πρέπει να προστίθεται η επιβεβαίωση αυτή.
- Η επικυρωμένη εγκαθίδρυση κλειδιού δεν επιτυγχάνεται εύκολα και δεν είναι καθόλου εύκολη εργασία να προσδιορίσεις και να διατυπώσεις με ακρίβεια τις απαιτήσεις ασφάλειας της.

Ενώ τα συμπεράσματα που προέκυψαν με τη βοήθεια της εφαρμογής και των διάφορων δοκιμών που έγιναν, συνίστανται στα εξής:

- Οι υπολογιστικές πτυχές του DHKE είναι αρκετά παρόμοιες με αυτές του RSA. Απαιτείται μήκος κλειδιού παρόμοιο με αυτό του RSA (1024 bit και άνω).
- Οι συμμετρικοί αλγόριθμοι είναι πολύ ταχύτεροι από τους ασύμμετρους, γι' αυτό το λόγο οι τελευταίοι χρησιμοποιούνται συνήθως στα διάφορα πρωτόκολλα αποκλειστικά και μόνο για την ανταλλαγή και ασφαλή εγκαθίδρυση του κλειδιού ενώ οι διαδικασίες κρυπτογράφησης / αποκρυπτογράφησης υλοποιούνται από τους πρώτους (AES, 3DES κ.α.)

Βιβλιογραφικές Αναφορές

[1] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, Scott Vanstone. *An Efficient Protocol for Authenticated Key Agreement*, Designs, Codes and Cryptography, 2003.

[2] Tatsuaki Okamoto. *Authenticated Key Exchange and Key Encapsulation in the Standard Model,* ASIACRYPT 2007, LNCS 4833, pp. 474–484, 2007.

[3] Simon Blake-Wilson, Alfred Menezes. *Authenticated Diffie-Hellman Key Agreement Protocols*, SAC'98, LNCS 1556, pp. 339 - 361, 1999.

[4] Michel Abdalla, David Pointcheval. *Simple Password-Based Encrypted Key Exchange Protocols,* CT-RSA 2005, LNCS 3376, pp. 191–208, 2005.

[5] Richard Schroeppel, Hilarie Orman, Sean O'Malley, Oliver Spatscheck. *Fast Key Exchange with Elliptic Curve Systems*. Advances in Cryptology - CRYPT0 '95, LNCS 963, pp. 43-56, 1995.

[6] W. Diffie, M. Hellman. *New Directions in Cryptography*, IEEE Transactions on Information Theory, Vol. IT-22, No. 6, pp. 644-654, 1976.

[7] N. Biggs. *Discrete Mathematics*. Oxford University Press, New York, 2nd edition, 2002.

[8] RSA Laboratories, PKCS#1: RSA Encryption Standard, version 1.5, 1993.

[9] D. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982.

[10] Victor Boyko, Philip MacKenzie, Sarvar Patel, *Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman*, EUROCRYPT 2000, LNCS 1807, pp. 156-171, 2000.

[11] N. Koblitz, A. Menezes, S. Vanstone. *The state of elliptic curve cryptography*, Designs, Codes and Cryptography, 19 (2000), 173-193.

[12] N. Koblitz. *A course in number theory and cryptography*, Springer-Verlag, New York (1994).

[13] ANSI X9.62-1999. *The Elliptic Curve Digital Signature Algorithm (ECDSA)*. Technical report, American Bankers Association, 1999.

[14] Ian F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, New York, NY, USA, 1999.

[15] E. Bach And J. Shallit. *Algorithmic Number Theory*, Volume I: Efficient Algorithms, MIT Press, Cambridge, Massachusetts, 1996.

[16] M. Bellare, R. Canetti, H. Krawczyk, *Keying hash functions for message authentication*, Advances in Cryptology–CRYPTO '96 (LNCS 1109), 1–15, 1996.

[17] L. GONG, *Using one-way functions for authentication*, Computer Communication Review, 19 (1989), 8–11.

[18] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Online textbook, 2004. Available at http://shoup.net/ntb.

[19] H. Cohen, G. Frey, and R. Avanzi. *Handbook of Elliptic and Hyperelliptic Curve Cryptography. Discrete Mathematics and Its Applications*. Chapman and Hall/CRC, September 2005.

[20] Dan Boneh, Ron Rivest, Adi Shamir, Len Adleman. *Twenty Years of Attacks on the RSA Cryptosystem*. Notices of the AMS, 46:203–213, 1999.

[21] D. Harkins, D. Carrel. *The Internet Key Exchange (IKE)*, RFC2409, 1998

[22] S. Kent, R. Atkinson. *IP Authentication Header*, RFC2402, 1998

[23] S. Kent, R. Atkinson. *IP Encapsulating Security Payload*, RFC2406, 1998

[24] BlueKrypt | Cryptographic Key Length Recommendation. [online] available from: <http://www.keylength.com/> [accessed 10/01/2013]

[25] G.J. Popek and C.S. Kline. *Encryption and Secure Computer Networks*, ACM ComputingSurveys, v. 11, n. 4, Dec 1979, pp. 331-356.

[26] ISO. Information Technology-Security Techniques-Key Management-Part 2: *Mechanisms using symmetric techniques*. ISO/IEC 11770-2, 1996.

[27] R.C. Merkle. *Secure Communication Over Insecure Channels*, *Communications of the ACM*, v.21, n. 4, 1978, pp. 294-299.

[28] ISO. Information Technology - Security Techniques - Key Management - Part 3: *Mechanisms using asymmetric techniques*. Technical Report ISO/IEC 11770-3, 1999.

[29] R.M. Needham and M.D. Schroeder. *Using Encryption for Authentication in Large Networks of Computers*, Communications of the ACM, v. 21, n. 12, Dec 1978, pp. 993-999.

[30] Alfred Menezes, Paul Van Oorschot, Scott Vastone. *Handbook of Applied Cryptography*,1997.

[31] Βασίλειος Κάτος, Γεώργιος Στεφανίδης.*Τεχνικές Κρυπτογρφίας & Κρυπτανάλυσης* ,2003.

[32] Πομπρότης Ανδρέας, Παπαδημητρίου Γεώργιος.*Ασφάλεια Δικτύων Υπολογιστών* ,2003.

[33] ΘΑΝΑΣΗΣ Π. ΞΕΝΟΣ. *Θεωρία Αριθμών,2010*.

[34] Joseph P. Buhler, Peter Stevenhage. *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography*.

Γλωσσάρι

Α		
	Αριθμός ακολουθίας	Sequence number
	Αποτύπωμα μηνύματος	Fingerprint of a message
Δ		
	Διανομή κλειδιού	Key distribution
Ε		
	Ενθυλάκωση ωφέλιμου φορτίου ασφάλειας	Encapsulating Security Payload - ESP
	Επιβεβαίωση κλειδιού	Key confirmation
	Εγκαθίδρυση κλειδιού	Key establishment
Κ		
	Κώδικας ελέγχου ταυτότητας μηνύματος	Message Authentication Code - MAC
	Κλειδί συνόδου	Session key
Μ		
	Μηχανισμός ενθυλάκωσης κλειδιού	Key Encapsulation Mechanism - KEM
	Μονόδρομη συνάρτηση	One-way function
	Μεταφορά κλειδιού	Key transport
Π		
	Πολιτική Ασφάλειας Δεδομένων	Security Policy Database - SPD
	Πρόβλημα διακριτού λογαρίθμου	Discrete logarithm problem - DLP
	Πιστοποίηση κλειδιού	Key authentication
Σ		
	Συνάρτηση κατακερματισμού	Hash function
	Συμφωνία πιστοποιημένου κλειδιού	Authenticated key agreement - AK
	Συμφωνία πιστοποιημένου κλειδιού με επιβεβαίωση κλειδιού	Authenticated key agreement with key confirmation - AKC
	Συμφωνία κλειδιού	Key agreement
	Σύνοψη μηνύματος	Digest

Παράρτημα Α

Κώδικας Εφαρμογής ClientServerCryptography

```
// Κώδικας του αρχείου ClientUI.java
import javax.swing.*;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.security.*;

import javax.crypto.*;

import java.security.interfaces.*;
import java.security.spec.X509EncodedKeySpec;
import java.math.BigInteger;

class ClientUI extends JFrame
{
    private static final long serialVersionUID = 1L;
    private JPanel connectPanel;
    private JTextField ipAddress;
    private JPanel msgShowPanel;
    private JTextArea msgShowArea;
    private JPanel msgEditPanel;
    private JTextArea msgEditArea;

    private Socket clientMsg = null, clientRSA = null, clientDES = null,
        clientDH = null;
    private static final int msgPort = 11268;
    private static final int RSAPort = 11234;
    private static final int DESPort = 11233;
    private static final int DHPort = 11269;
    private ObjectInputStream ObjISMsg;
    private ObjectOutputStream ObjOSMsg;

    private RSA clientKeyRSA;
    private DH paramsDH;
    private PublicKey serverPBK;
    private SecretKey keyDES;
    private SecretKey clientDesKeyDH;

    private Font font = new Font("Dialog",Font.BOLD,12);

    private String file;
    private String algorithm;

    private Long startTimeCreateKeys;
    private Long endTimeCreateKeys;
    private Long totalTimeCreateKeys;

    public ClientUI(String file, String algorithm) throws IOException
    {
        super("Client "+algorithm.toUpperCase()+" encryption");
        this.file = file;
        this.algorithm = algorithm;
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Αρχικοποίηση διεπαφής
        initConnectPanel();
        initMsgShowPanel();
```

```
        initMsgEditPanel();

        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());
        pane.add(connectPanel,BorderLayout.NORTH);
        pane.add(msgShowPanel,BorderLayout.CENTER);
        pane.add(msgEditPanel,BorderLayout.SOUTH);

        pack();
        setVisible(true);
        // Ανάλογα την παράμετρο algorithm που περνάμε στην εφαρμογή
        // δημιουργείται και το αντίστοιχο στιγμιότυπο παραμέτρων πρωτοκόλλου
        try
        {
            startTimeCreateKeys = System.currentTimeMillis();
            if (algorithm.equalsIgnoreCase("dh"))
            {
                // Δημιουργία παραμέτρων για το πρωτόκολλο DH
                paramsDH = new DH();
            }
            else
            {
                // Δημιουργία παραμέτρων για το πρωτόκολλο RSA
                clientKeyRSA = new RSA();
            }
            endTimeCreateKeys = System.currentTimeMillis();
            totalTimeCreateKeys = endTimeCreateKeys - startTimeCreateKeys;
        }
        catch(Exception e)
        {
           System.out.println("e");
        }
    }
    // Δημιουργείται το JPanel που «φιλοξενεί» το πεδίο της ip διεύθυνσης του server
    // (μηχανή) στον οποίο εκτελείται η εφαρμογή μας και το κουμπί με το οποίο θα
    // ζητήσουμε τη σύνδεση
    private void initConnectPanel()
    {
        connectPanel = new JPanel();
        connectPanel.setLayout(new FlowLayout());

        JButton connectButton = new JButton("Σύνδεση");
        connectButton.setFont(font);
        connectButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
            {
                connectServer(ipAddress.getText());
            }
        });

        ipAddress = new JTextField(10);
        ipAddress.setText("127.0.0.1");

        connectPanel.add(connectButton);
        connectPanel.add(ipAddress);
    }
    /* Επιλέξαμε ανάλογα με τη τιμή της παραμέτρου algorithm να δημιουργείται ένα
       socket σε διαφορετική θύρα, έτσι ώστε να είναι πιο ευανάγνωστος ο κώδικας.
       Σε κάθε περίπτωση δημιουργείται ένα socket για την αποστολή του καθαρού
       μηνύματος και ένα για την ανταλλαγή κλειδιών DES.
    */
    private void connectServer(String serverAddress)
    {
```

```
        try
        {
            clientMsg = new Socket(serverAddress,msgPort);
            clientDES = new Socket(serverAddress,DESPort);

            if (algorithm.equalsIgnoreCase("dh"))
            {
               clientDH = new Socket(serverAddress,DHPort);
               /* Στη περίπτωση που έχουμε επιτυχή σύνδεση των socket στις εκάστοτε
                  θύρες, τότε δημιουργούνται και εκκινούν δύο νήματα. Το ένα για τη
                  λειτουργία του πρωτοκόλλου και το αλλό για το κρυπτογραφημένο
                  μήνυμα
               */
               if(clientMsg.isBound() == true && clientDH.isBound() == true &&
                 clientDES.isBound() == true)
               {
                   msgShowArea.append("\n**Επιτυχής Σύνδεση\n");
                   new keyThreadDH(clientDH, clientDES).start();
                   new recThread(clientMsg).start();
               }
                else
                {
                   msgShowArea.append("\n**Αποτυχής Σύνδεση");
                }
            }
            else
            {
                clientRSA = new Socket(serverAddress,RSAPort);
               /* Στη περίπτωση που έχουμε επιτυχή σύνδεση των socket στις εκάστοτε
                  θύρες, τότε δημιουργούνται και εκκινούν δύο νήματα. Το ένα για τη
                  λειτουργία του πρωτοκόλλου και το αλλό για το κρυπτογραφημένο
                  μήνυμα
               */
                if(clientMsg.isBound() == true && clientRSA.isBound() == true &&
                     clientDES.isBound() == true)
                {
                    msgShowArea.append("\n**Επιτυχής Σύνδεση\n");
                    new keyThread(clientRSA, clientDES).start();
                    new recThread(clientMsg).start();
                }
                else
                {
                   msgShowArea.append("\n**Αποτυχής Σύνδεση");
                }
            }
        }
        catch(Exception e)
        {
           System.out.println(e);
        }
    }
    /*
     * Η κλάση που ακολουθεί είναι νήμα και έχει πέντε πεδία. Τα τρία είναι τύπου
     * ObjectInputStream και ObjectOutputStream τα οποία εκτός των άλλων, δίνουν τη
     * δυνατότητα ανάμεσα σε hosts μέσω των sockets να ανταλλάσουν objects. Έτσι στη
     * περίπτωση της εφαρμογής μας γίνεται η εγγραφή και η ανάγνωση των objects που
     * αφορούν την ανταλλαγή και την εγκαθίδρυση των κλειδιών των πρωτοκόλλων μας.
     */

    class keyThread extends Thread
    {
        private ObjectInputStream ObjISDES;
        private ObjectInputStream ObjISRSA;
```

```
        private ObjectOutputStream ObjOSRSA;

        byte[] bkeyDES;
/*
 * Η κλάση Cipher παρέχει τη δυνατότητα ενός κρυπτοκώδικα για κρυπτογράφηση /
 * αποκρυπτογράφηση.
 */
        Cipher cipher;
/*
 * Ο γεννήτορας του νήματος δέχεται δύο παραμέτρους τύπου Socket. Ένα για τον
 * RSA κι ένα για τον DES. Προσοχή! Για τον RSA υπάρχουν δύο σενάρια στην
 * εφαρμογή μας. Πρώτο σενάριο να ανταλλάξουν οι hosts το μυστικό κλειδί του
 * DES μέσω του RSA ώστε να γίνει κρυπτογράφηση/αποκρυπτογράφηση με αυτό το
 * κλειδί και το άλλο σενάριο να γίνει η κρυπτογράφηση/αποκρυπτογράφηση με το
 * δημόσιο/ιδιωτικό κλειδί αντίστοιχα του RSA. Σε κάθε περίπτωση εμείς
 * δημιουργούμε πάντα ObjOSRSA και ObjISRSA για να μη μπαίνουμε στη διαδικασία
 * να ελέγχουμε τη διάλεξε ο χρήστης να εκτελέσει. Σημειώνουμε ότι το ObjISRSA
 * χρησιμοποιείται στη περίπτωση που η κρυπτογράφηση/αποκρυπτογράφηση θα γίνει με το
 * δημόσιο/ιδιωτικό κλειδί αντίστοιχα του RSA.
 */
        public keyThread(Socket sRSA, Socket sDES) throws IOException,
NoSuchAlgorithmException, InvalidAlgorithmParameterException, InvalidKeyException
        {
            ObjOSRSA = new ObjectOutputStream(sRSA.getOutputStream());
            ObjISRSA = new ObjectInputStream(sRSA.getInputStream());
            ObjISDES = new ObjectInputStream(sDES.getInputStream());
 /*
  * Υπενθυμίζουμε ότι στο constructor του ClientUI δημιουργούμε στις περιπτώσεις που
  * δεν πρόκειται να υλοποιηθεί dh ένα στιγμιότυπο του RSA, για να έχουμε τους
  * παραμέτρους του. Στο αρχείο RSA.java θα εξηγήσουμε αναλυτικά τον τρόπο που
  * δημιουργούνται και χρησιμοποιούνται. Γράφουμε λοιπόν στο ObjectOutputStream το
  * δημόσιο κλειδί του RSA.
  */
            ObjOSRSA.writeObject(clientKeyRSA.getPublicKey());
            ObjOSRSA.flush();

            try
            {
               // Έναρξη χρόνου διαδικασίας δημιουργίας κατάλληλων κλειδιών.
               startTimeCreateKeys = System.currentTimeMillis();
               /*
                * Η getInstance επιστρέφει ένα αντικείμενο που υλοποιεί τον
               * μετασχηματισμό RSA στη περίπτωση μας. Σύμφωνα με τη JCA API η Cipher
               * υποστηρίζει και τον RSA.
                */
               cipher = Cipher.getInstance("RSA");
               /* Στη περίπτωση που ο χρήστης επιλέξει να εκτελέσει το σενάριο με το
               * DES, τότε το στιγμιότυπο του cipher αρχικοποιείται με το ιδιωτικό
               * κλειδί του RSA με το οποίο θα κρυπτογραφηθεί το μυστικό κλειδί DES
               * για να σταλθεί με ασφάλεια στον server, ώστε στη συνέχεια να
               * μπορέσει εκείνος να το αποκρυπτογραφήσει με το ιδιωτικό κλειδί του
               * RSA και να το χρησιμοποιήσει για να αποκρυπτογραφήσει το
               * κρυπτογραφημένο μήνυμα που θα λάβει από τον client. Το UNWRAP_MODE
               * «μετατρέπει» το μυστικό κλειδί του RSA σε java.security.Key
               * αντικείμενο.
               */
               if(algorithm.equalsIgnoreCase("des"))
                   cipher.init(Cipher.UNWRAP_MODE, clientKeyRSA.getPrivateKey());
               // Λήξη χρόνου διαδικασίας δημιουργίας κατάλληλων κλειδιών.
               endTimeCreateKeys = System.currentTimeMillis();
               totalTimeCreateKeys = totalTimeCreateKeys + (endTimeCreateKeys -
                                                     startTimeCreateKeys);
            }
```

```
            catch(Exception e)
            {
               System.out.println("Cipher:"+e);
            }
        }
/*
 * Η κλήση της run() γίνεται όταν δημιουργείται το νέο thread με τη κλήση start().
 * Στη περίπτωση μας γίνεται στη μέθοδο connectServer.
 */
        public void run()
        {
            try
            {
                 /* Ο client διαβάζει το δημόσιο κλειδί του server. Γιατί αυτό το
                   thread δημιουργείται στη περίπτωση που ο χρήστης εκτελέσει des ή
                   rsa. Και στις δύο περιπτώσεις θα πρέπει να κρυπτογραφήσουμε με το
                   δημόσιο κλειδί του server ώστε να μπορέσει εκείνος να
                   αποκρυπτογραφήσει με το ιδιωτικό του.
                 */
                serverPBK = (RSAPublicKey) ObjISRSA.readObject();
                ObjOSRSA.close();
                ObjISRSA.close();

                /* Στη συνέχεια διαβάζει από το Input stream το μυστικό
                   κλειδί του DES με το οποίο κρυπτογραφεί και αποκρυπτογραφεί
                   τα μηνύματα.
                */
                if(algorithm.equalsIgnoreCase("des"))
                {
                   bkeyDES = (byte[]) ObjISDES.readObject();

                   startTimeCreateKeys = System.currentTimeMillis();
                  // Δημιουργούμε από το object που διαβάσαμε προηγουμένως το μυστικό
                  // κλειδί DES.
                   keyDES = (SecretKey) cipher.unwrap(bkeyDES,
                                                      "DES",Cipher.SECRET_KEY);
                  endTimeCreateKeys = System.currentTimeMillis();
                  totalTimeCreateKeys = totalTimeCreateKeys + (endTimeCreateKeys
                                                        - startTimeCreateKeys);
                   ObjISDES.close();

                   msgShowArea.append("Client RSA private
key:"+byteToBinary(clientKeyRSA.getPrivateKey().getEncoded())+"\n");
                    msgShowArea.append("Client RSA public
key:"+byteToBinary(clientKeyRSA.getPublicKey().getEncoded())+"\n");
                    msgShowArea.append("Server RSA public
key:"+byteToBinary(serverPBK.getEncoded())+"\n");
                    msgShowArea.append("DES secret
key:"+byteToBinary(keyDES.getEncoded()));
                    msgShowArea.append("\n**Ο συνολικός χρόνος παραγωγής και
συμφωνίας του μυστικού κλειδιού DES που θα χρησιμοποιηθεί για τη
\nκρυπτογράφηση/αποκρυπτογράφηση μέσω του πρωτοκόλλου RSA διήρκησε:
"+totalTimeCreateKeys+" χιλιοστά του δευτερολέπτου.");
                    msgShowArea.setCaretPosition(msgShowArea.getText().length());
                }
                // Στη περίπτωση που πρόκειται για τον RSA, το κλειδί της
                // κρυπτογράφησης είναι το ίδιο με αυτό που εγκαθιδρύεται!
                else if(algorithm.equalsIgnoreCase("rsa"))
                {
                   msgShowArea.append("Client RSA private
key:"+byteToBinary(clientKeyRSA.getPrivateKey().getEncoded())+"\n");
                    msgShowArea.append("Client RSA public
key:"+byteToBinary(clientKeyRSA.getPublicKey().getEncoded())+"\n");
```

```
                    msgShowArea.append("Server RSA public
key:"+byteToBinary(serverPBK.getEncoded())+"\n");
                    msgShowArea.append("\n**Ο συνολικός χρόνος παραγωγής και του
ζεύγους κλειδιών RSA που θα χρησιμοποιηθούν για τη \nκρυπτογράφηση/αποκρυπτογράφηση
μέσω του πρωτοκόλλου RSA διήρκησε: "+totalTimeCreateKeys+" χιλιοστά του
δευτερολέπτου.");
                    msgShowArea.setCaretPosition(msgShowArea.getText().length());
                }
            }
            catch(Exception e)
            {
               System.out.println(e);
            }
        }
    }
    /* Το νήμα που δημιουργείται στη περίπτωση του DH. Στη κλάση αυτή έχουμε ένα
       πεδίο τύπου KeyAgreement, που είναι μια κλάση που παρέχει τη δυνατότητα
       συμφωνίας ή ανταλλαγής κλειδιού σε πρωτόκολλα κρυπτογράφησης.
     */
    class keyThreadDH extends Thread
    {
        private ObjectInputStream ObjISDES;
        private ObjectInputStream ObjISDH;
        private ObjectOutputStream ObjOSDH;

        private KeyAgreement keyAgreeDH;
        byte[] clientPubKeyEncDH;

        // Δημιουργία αντίστοιχων streams με τους RSA και DES
        public keyThreadDH(Socket sDH, Socket sDES) throws IOException,
NoSuchAlgorithmException, InvalidAlgorithmParameterException, InvalidKeyException
        {
           ObjOSDH = new ObjectOutputStream(sDH.getOutputStream());
           ObjISDH = new ObjectInputStream(sDH.getInputStream());
           ObjISDES = new ObjectInputStream(sDES.getInputStream());
        }

        public void run()
        {
            try
            {
               // Στιγμή εκκίνησης συμφωνίας κλειδιών
               startTimeCreateKeys = System.currentTimeMillis();
               // Παράγεται το ζεύγος κλειδιών του DH για τον client
            KeyPairGenerator clientKpairGen = KeyPairGenerator.getInstance("DH");
               // Αρχικοποιείται ο generator χρησιμοποιώντας τις παραμέτρους του DH
            clientKpairGen.initialize(paramsDH.getDhSkipParamSpec());
            KeyPair clientKpair = clientKpairGen.generateKeyPair();

            // Ο client δημιουργεί και αρχικοποιεί το αντικείμενο KeyAgreement
            keyAgreeDH = KeyAgreement.getInstance("DH");
            keyAgreeDH.init(clientKpair.getPrivate());

            // Ο client κωδικοποιεί το δημόσιο κλειδί του και το στέλνει στο server.
            clientPubKeyEncDH = clientKpair.getPublic().getEncoded();

            endTimeCreateKeys = System.currentTimeMillis();
            totalTimeCreateKeys += (endTimeCreateKeys - startTimeCreateKeys);

                ObjOSDH.writeObject(clientPubKeyEncDH);
                ObjOSDH.flush();

               byte[] serverPubKeyEncDH = (byte[]) ObjISDH.readObject();
```

```
                // Ο client χρησιμοποιεί το δημόσιο κλειδί για τη πρώτη φάση
                // για τη δική του έκδοση του πρωτοκόλλου DH. Πριν προλάβει να
                // το κάνει θα πρέπει να αρχικοποιήσει ένα δημόσιο κλειδί DH από
                // το κωδικοποιημένο υλικό κλειδιού του server
                startTimeCreateKeys = System.currentTimeMillis();
                KeyFactory clientKeyFac = KeyFactory.getInstance("DH");

        X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(serverPubKeyEncDH);
        PublicKey serverPubKeyDH = clientKeyFac.generatePublic(x509KeySpec);
        keyAgreeDH.doPhase(serverPubKeyDH, true);

                 byte[] clientSharedSecret = keyAgreeDH.generateSecret();
                 endTimeCreateKeys = System.currentTimeMillis();
                totalTimeCreateKeys += (endTimeCreateKeys - startTimeCreateKeys);

                 System.out.println("Client secret (Hex format): " +
toHexString(clientSharedSecret)+"\nClient secret (Binary
format):"+byteToBinary(clientSharedSecret));
                int clientLen = clientSharedSecret.length;

                ObjOSDH.flush();
                ObjOSDH.writeObject(clientLen);

                startTimeCreateKeys = System.currentTimeMillis();
                    keyAgreeDH.doPhase(serverPubKeyDH, true);
                    clientDesKeyDH = keyAgreeDH.generateSecret("DES");
                endTimeCreateKeys = System.currentTimeMillis();
                totalTimeCreateKeys += (endTimeCreateKeys - startTimeCreateKeys);

                 msgShowArea.append("Client DH private
key:"+byteToBinary(clientSharedSecret)+"\n");
                 msgShowArea.append("Client DH public
key:"+byteToBinary(clientPubKeyEncDH)+"\n");
                 msgShowArea.append("Server DH public
key:"+byteToBinary(serverPubKeyEncDH)+"\n");
                 msgShowArea.append("DES secret
key:"+byteToBinary(clientDesKeyDH.getEncoded()));
                 msgShowArea.append("\n**Ο συνολικός χρόνος παραγωγής και συμφωνίας
του μυστικού κλειδιού DES που θα χρησιμοποιηθεί για τη
\nκρυπτογράφηση/αποκρυπτογράφηση μέσω του πρωτοκόλλου DH διήρκησε:
"+totalTimeCreateKeys+" χιλιοστά του δευτερολέπτου.");
                 msgShowArea.setCaretPosition(msgShowArea.getText().length());
            }
            catch(Exception e)
            {
               System.out.println(e);
            }
        }
    }
    // Η κλάση αυτή αποτελεί ένα νήμα που καταγράφει στο panel του client τα μηνύματα
    // αποστολής και το ιστορικό αυτών.
    class recThread extends Thread
    {
        private byte[] eData;

        public recThread(Socket sMsg) throws IOException
        {
            ObjOSMsg=new ObjectOutputStream(sMsg.getOutputStream());
            ObjISMsg=new ObjectInputStream(sMsg.getInputStream());
        }

        public void run()
```

```
        {
            try
            {
                while(true)
                {
                   eData = (byte[]) ObjISMsg.readObject();
                }
            }
            catch(Exception e)
            {
               System.out.println(e);
            }
        }
    }

    private void initMsgShowPanel()
    {
        msgShowPanel = new JPanel();
        msgShowPanel.setLayout(new BorderLayout());

        JLabel label = new JLabel("Μηνύματα Διαδικασίας: ");
        label.setFont(font);

        msgShowArea = new JTextArea(10,80);
        msgShowArea.setEditable(false);
        JScrollPane msgShowPane = new JScrollPane();
        msgShowPane.setViewportView(msgShowArea);

        msgShowPanel.add(label,BorderLayout.NORTH);
        msgShowPanel.add(msgShowPane,BorderLayout.CENTER);
    }

    private void initMsgEditPanel() throws IOException
    {
        msgEditPanel = new JPanel();
        msgEditPanel.setLayout(new BorderLayout());

        //Αρχικοποιούμε το μήνυμα με ένα αρκετά μεγάλο κείμενο ώστε να
        //γίνει αισθητή η διαφορά μεταξύ των αλγορίθμων.
        StringBuffer strBuf = new StringBuffer();
        try
        {
           FileInputStream fs = new FileInputStream(new File(file));
            byte[] data = new byte[fs.available()];
            int i=0;
            while ((i = fs.read(data)) != -1)
            {
               strBuf.append(new String(data, 0, i));
            }
            fs.close();
        }catch(Exception b)
        {
           System.out.println(b);
        }

        JLabel label = new JLabel("Κείμενο προς κρυπτογράφηση και αποστολή: ");
        label.setFont(font);

        msgShowArea.append("**Μέγεθος μηνύματος προς κρυπτογράφηση και αποστολή του
αρχείου \""+file+"\": "+Integer.toString(strBuf.length())+" χαρακτήρες.");
        if (algorithm.equalsIgnoreCase("dh"))
           msgShowArea.append("\n**Η δημιουργία των Diffie-Hellman παραμέτρων
καταναλώνει αρκετό χρόνο..." +
```

```
                    "\n**Παρακαλώ περιμένετε την εκκίνηση του Server...");
            msgEditArea = new JTextArea(5,80);
            msgEditArea.append(strBuf.toString());

            JScrollPane msgEditPane = new JScrollPane();
            msgEditPane.setViewportView(msgEditArea);

            JPanel buttonPanel = new JPanel();

            JButton sendButtonDES = new JButton("Αποστολή (RSA-DES)");
            sendButtonDES.setToolTipText("Κρυπτογράφηση κλειδιού DES μέσω RSA.
Κρυπτογράφηση/Αποκρυπτογράφηση μέσω DES.");

            JButton sendButtonRSA = new JButton("Αποστολή (RSA-RSA)");
            sendButtonRSA.setToolTipText("Κρυπτογράφηση/Αποκρυπτογράφηση μέσω RSA.");

            JButton sendButtonDH = new JButton("Αποστολή (DH-DES)");
            sendButtonDH.setToolTipText("Κρυπτογράφηση μέσω εγκαθίδρυσης κλειδιού DH.
Κρυπτογράφηση/Αποκρυπτογράφηση μέσω DES.");

            JButton reloadButton = new JButton("Επαναφόρτωση Κειμένου");

            sendButtonDES.setFont(font);
            sendButtonRSA.setFont(font);
            sendButtonDH.setFont(font);
            reloadButton.setFont(font);

            reloadButton.addActionListener(new ActionListener() {
               public void actionPerformed(ActionEvent e) {
                   try
                   {
                         FileInputStream fs = new FileInputStream(new File(file));
                        StringBuffer strBuf = new StringBuffer();

                        byte[] data = new byte[fs.available()];
                        int i=0;
                        while ((i = fs.read(data)) != -1)
                        {
                          strBuf.append(new String(data, 0, i));
                        }
                        fs.close();

                        msgEditArea.append(strBuf.toString());
                   }
                   catch (Exception b)
                   {
                        System.out.println(b);
                    }
               }
            });

            sendButtonDES.addActionListener(new ActionListener(){
              public void actionPerformed(ActionEvent e)
              {
                  try
                  {
                     // Ελέγχουμε αν έχει γίνει η σύνδεση με τον server και αν ο
                     // αλγόριθμος που έχει επιλεχθεί είναι ο rsa-des.
                     if(clientDES != null && algorithm.equalsIgnoreCase("des"))
                      {
                        Long startMilliTime = System.currentTimeMillis();
                        msgShowArea.append("\n\n-Η κρυπτογράφηση (DES) ξεκίνησε στις:
"+startMilliTime);
```

```
                        byte[] data = msgEditArea.getText().getBytes();
                        // Κρυπτογράφηση μηνύματος με το μυστικό κλειδί του DES
                        // που έχει διανεμηθεί ασφαλώς μέσω του ασύμμετρου
                        // πρωτοκόλλου RSA
                        byte[] eData = DES.SEnc(keyDES, "ENC", data);

                        Long finishMilliTime = System.currentTimeMillis();
                        msgShowArea.append("\n-Η κρυπτογράφηση (DES) έληξε στις:
"+finishMilliTime);
                        msgShowArea.append("\n-Διάρκεια: "+(finishMilliTime-
startMilliTime)+" milliseconds.");
                        msgShowArea.append("\n-Μέγεθος καθαρού μηνύματος:
"+data.length+" χαρακτήρες.");
                        msgShowArea.append("\n-Μέγεθος κρυπτογραφημένου μηνύματος:
"+eData.length+" χαρακτήρες.");
                        msgShowArea.append("\n-Αύξηση μεγέθους κατά:
"+(((100.0/data.length)*eData.length)-100)+"%.");
                        ObjOSMsg.writeObject(eData);
                        ObjOSMsg.flush();

                        msgShowArea.setCaretPosition(msgShowArea.getText().length());
                        msgEditArea.setText(null);
                      }
                    else
                    {
                        msgShowArea.append("\n\n**Προσοχή!Θα πρέπει πρώτα να
συνδεθείτε στον server με mode κρυπτογράφησης DES.");
                    }
                }
                catch (Exception b)
                {
                   System.out.println(b);
                }
            }
        });

        sendButtonRSA.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    // Ελέγχουμε αν έχει γίνει η σύνδεση με τον server και αν ο
                    // αλγόριθμος που έχει επιλεχθεί είναι ο rsa-rsa.
                    if(clientRSA != null && algorithm.equalsIgnoreCase("rsa"))
                    {
                        Long startMilliTime = System.currentTimeMillis();
                        msgShowArea.append("\n\n-Η κρυπτογράφηση (RSA) ξεκίνησε στις:
"+startMilliTime);

                        String data = msgEditArea.getText();
                        // Κωδικοποίηση μηνύματος με το δημόσιο κλειδί του RSA.
                        byte[] eData = clientKeyRSA.encryptRSA(serverPBK, data);

                        Long finishMilliTime = System.currentTimeMillis();
                        msgShowArea.append("\n-Η κρυπτογράφηση (RSA) έληξε στις:
"+finishMilliTime);
                        msgShowArea.append("\n-Διάρκεια: "+(finishMilliTime-
startMilliTime)+" milliseconds.");
                        msgShowArea.append("\n-Μέγεθος καθαρού μηνύματος:
"+data.length()+" χαρακτήρες.");
                        msgShowArea.append("\n-Μέγεθος κρυπτογραφημένου μηνύματος:
"+eData.length+" χαρακτήρες.");
```

```
                        msgShowArea.append("\n-Αύξηση μεγέθους κατά:
"+(((100.0/data.length())*eData.length)-100)+"%.");

                        ObjOSMsg.writeObject(eData);
                        ObjOSMsg.flush();

                        msgShowArea.setCaretPosition(msgShowArea.getText().length());
                        msgEditArea.setText(null);
                     }
                    else
                    {
                        msgShowArea.append("\n\n**Προσοχή!Θα πρέπει πρώτα να
συνδεθείτε στον server με mode κρυπτογράφησης RSA.");
                    }
                }
                catch (Exception b)
                {
                   System.out.println(b);
                }
            }
        });

        sendButtonDH.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    // Ελέγχουμε αν έχει γίνει η σύνδεση με τον server και αν ο
                   // αλγόριθμος που έχει επιλεχθεί είναι ο dh-des.
                    if(clientDH != null && algorithm.equalsIgnoreCase("dh"))
                     {
                      Long startMilliTime = System.currentTimeMillis();
                      msgShowArea.append("\n\n-Η κρυπτογράφηση (DH) ξεκίνησε στις:
"+startMilliTime);

                       byte[] data = msgEditArea.getText().getBytes();
                       // Κωδικοποίηση μηνύματος με το μυστικό κλειδί του DES
                       // που έχει διανεμηθεί ασφαλώς μέσω του ασύμμετρου
                       // πρωτοκόλλου Diffie-Hellman.
                       byte[] eData = DES.SEnc(clientDesKeyDH, "ENC", data);

                       Long finishMilliTime = System.currentTimeMillis();
                       msgShowArea.append("\n-Η κρυπτογράφηση (DH) έληξε στις:
"+finishMilliTime);
                       msgShowArea.append("\n-Διάρκεια: "+(finishMilliTime-
startMilliTime)+" milliseconds.");
                       msgShowArea.append("\n-Μέγεθος καθαρού μηνύματος:
"+data.length+" χαρακτήρες.");
                       msgShowArea.append("\n-Μέγεθος κρυπτογραφημένου μηνύματος:
"+eData.length+" χαρακτήρες.");
                       msgShowArea.append("\n-Αύξηση μεγέθους κατά:
"+(((100.0/data.length)*eData.length)-100)+"%.");

                       ObjOSMsg.writeObject(eData);
                       ObjOSMsg.flush();

                       msgShowArea.setCaretPosition(msgShowArea.getText().length());
                       msgEditArea.setText(null);
                     }
                    else
                    {
                        msgShowArea.append("\n\n**Προσοχή!Θα πρέπει πρώτα να
συνδεθείτε στον server με mode κρυπτογράφησης DH.");
```

```
                    }
                }
                catch (Exception b)
                {
                   System.out.println(b);
                }
            }
        });

        buttonPanel.add(reloadButton);
        if (algorithm.equalsIgnoreCase("des"))
           buttonPanel.add(sendButtonDES);
        else if (algorithm.equalsIgnoreCase("rsa"))
           buttonPanel.add(sendButtonRSA);
        else if (algorithm.equalsIgnoreCase("dh"))
           buttonPanel.add(sendButtonDH);

        msgEditPanel.add(label,BorderLayout.NORTH);
        msgEditPanel.add(msgEditPane,BorderLayout.CENTER);
        msgEditPanel.add(buttonPanel,BorderLayout.SOUTH);
    }

    // Επιστρέφει σε δυαδική μορφή (τύπου String) το περιεχόμενο του buffer
    private static String byteToBinary (byte[] bytes)
    {
        BigInteger bi = new BigInteger(bytes);
        return bi.toString(2);
    }

    // Επιστρέφει σε δεκαεξαδική μορφή (τύπου String) το περιεχόμενο του buffer
    private void byte2hex(byte b, StringBuffer buf)
    {
     char[] hexChars = { '0', '1', '2', '3', '4', '5', '6', '7', '8',
     '9', 'A', 'B', 'C', 'D', 'E', 'F' };
     int high = ((b & 0xf0) >> 4);
     int low = (b & 0x0f);
     buf.append(hexChars[high]);
     buf.append(hexChars[low]);
    }

    // Μετατρέπει το μήνυμα από δεκαεξαδική μορφή σε String
    private String toHexString(byte[] block)
    {
    StringBuffer buf = new StringBuffer();

    int len = block.length;

        for (int i = 0; i < len; i++)
        {
            byte2hex(block[i], buf);
            if (i < len - 1)
            {
                buf.append(":");
            }
        }
        return buf.toString();
    }
 }
// Η κλάση tester περιέχει τη μέθοδο main για την εκκίνηση της εφαρμογής. Ταυτόχρονα
// με τον server, «σηκώνεται» και το στιγμιότυπο του client.
class tester
{
    public static void main(String[] args) throws IOException
```

```
    {
        if(args.length == 2)
            new ServerUI(args[0], args[1]);
        else
            System.out.println("You must type 2 arguments. First the name of the
algorithm and second the full filename of the cleartext.");
    }
}

// Κώδικας του αρχείου ServerUI.java
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.security.*;

import javax.crypto.*;
import javax.crypto.interfaces.DHPublicKey;
import javax.crypto.spec.DHParameterSpec;

import java.security.interfaces.*;
import java.security.spec.X509EncodedKeySpec;
import java.math.BigInteger;

class ServerUI extends JFrame
{
    private static final long serialVersionUID = 1L;
    private JPanel msgShowPanel;
    private JTextArea msgShowArea;

    private ServerSocket serverMsg, serverRSA, serverDES, serverDH;
    private static final int msgPort = 11268;
    private static final int RSAPort = 11234;
    private static final int DESPort = 11233;
    private static final int DHPort = 11269;
    private ObjectInputStream ObjISMsg;
    private ObjectOutputStream ObjOSMsg;

    private RSA serverKeyRSA;
    private PublicKey clientPBK;
    private SecretKey keyDES;
    private SecretKey serverDesKeyDH;

    private Font font = new Font("Dialog",Font.BOLD,18);

    private String algorithm;

    public ServerUI(String algorithm, String file)
    {
        super("Server "+algorithm.toUpperCase()+" decryption");
        try
        {
          this.algorithm = algorithm;
         // Εκκίνηση του client
          new ClientUI(file, algorithm);
       } catch (IOException e1) {
          e1.printStackTrace();
       }
```

```
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(0, 390);

        // Δημιουργία της διεπαφής του server
        initMsgShowPanel();

        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());
        pane.add(msgShowPanel,BorderLayout.CENTER);

        pack();
        setVisible(true);

        try
        {
          // Δημιουργούμε στιγμιότυπο του RSA για τις περιπτώσεις DES και RSA
          serverKeyRSA = new RSA();

          if(algorithm.equalsIgnoreCase("des"))
          {
               // Δημιουργία παταμέτρων DES
               DES DES = new DES();
               // Το μυστικό κλειδί του DES
               keyDES = DES.getSecretKey();
               // Δημιουργία των κατάλληλων sockets
               serverMsg = new ServerSocket(msgPort);
               serverRSA = new ServerSocket(RSAPort);
               serverDES = new ServerSocket(DESPort);

               msgShowArea.append("DES DECRYPTION BY SERVER:\n\n");
               // Εκκίνηση νημάτων αντίστοιχων με εκείνων του client
               new keyThread(serverRSA.accept(),
serverDES.accept()).start();
               new recThread(serverMsg.accept()).start();
          }
          else if(algorithm.equalsIgnoreCase("rsa"))
          {
             // Δημιουργία των κατάλληλων sockets
              serverMsg = new ServerSocket(msgPort);
              serverRSA = new ServerSocket(RSAPort);
              serverDES = new ServerSocket(DESPort);

               msgShowArea.append("RSA DECRYPTION BY SERVER:\n\n");
              // Εκκίνηση νημάτων αντίστοιχων με εκείνων του client
              new keyThread(serverRSA.accept(), serverDES.accept()).start();
              new recThread(serverMsg.accept()).start();
          }
          else if(algorithm.equalsIgnoreCase("dh"))
          {
           /* Στη περίπτωση του DH το μυστικό κλειδί προκύπτει από το υλικό
              του δημοσίου κλειδιού του DH, γι'αυτό δε δημιουργούμε
              αντικείμενο  DES. */
             serverMsg = new ServerSocket(msgPort);
             serverDH = new ServerSocket(DHPort);
             serverDES = new ServerSocket(DESPort);

             msgShowArea.append("DH DECRYPTION BY SERVER:\n\n");
```

```
                // Εκκίνηση των αντίστοιχων νημάτων για τη περίπτωση του DH
                new keyThreadDH(serverDH.accept(), serverDES.accept()).start();
                new recThread(serverMsg.accept()).start();
            }
        }
        catch(Exception e)
        {
          System.out.println("server.accept:"+e);
        }
    }

    class keyThread extends Thread
    {
        private ObjectOutputStream ObjOSDES;
        private ObjectOutputStream ObjOSRSA;
        private ObjectInputStream ObjISRSA;
        byte[] keyDESbyte;

        // Δημιουργία των streams στα αντίστοιχα sockets για τους RSA και DES
        public keyThread(Socket sRSA, Socket sDES) throws IOException
        {
            ObjOSDES = new ObjectOutputStream(sDES.getOutputStream());
            ObjOSRSA = new ObjectOutputStream(sRSA.getOutputStream());
            ObjISRSA = new ObjectInputStream(sRSA.getInputStream());
        }

        public void run()
        {
            try
            {
                // Ο server διαβάζει το δημόσιο κλειδί του client
                clientPBK = (RSAPublicKey) ObjISRSA.readObject();

                // Ο server γράφει το δημόσιο κλειδί του στο stream
                ObjOSRSA.writeObject(serverKeyRSA.getPublicKey());
                ObjOSRSA.flush();
                ObjOSRSA.close();
                ObjISRSA.close();

                if(algorithm.equalsIgnoreCase("des"))
                {
                // Κάνουμε wrap το μυστικό κλειδί DES με το δημόσιο RSA
                // κλειδί του client
                    keyDESbyte = RSA.wrapkey(keyDES, clientPBK);
                    ObjOSDES.writeObject(keyDESbyte);
                    ObjOSDES.flush();
                    ObjOSDES.close();

                    msgShowArea.append("Server RSA private
key:"+byteToBinary(serverKeyRSA.getPrivateKey().getEncoded())+"\n");
                    msgShowArea.append("Server RSA public
key:"+byteToBinary(serverKeyRSA.getPublicKey().getEncoded())+"\n");
                    msgShowArea.append("Client RSA public
key:"+byteToBinary(clientPBK.getEncoded())+"\n");
                    msgShowArea.append("DES secret
key:"+byteToBinary(keyDES.getEncoded())+"\n");
```

```
msgShowArea.setCaretPosition(msgShowArea.getText().length());
                }
                else if(algorithm.equalsIgnoreCase("rsa"))
                {
                 msgShowArea.append("Server RSA private
key:"+byteToBinary(serverKeyRSA.getPrivateKey().getEncoded())+"\n");
                    msgShowArea.append("Server RSA public
key:"+byteToBinary(serverKeyRSA.getPublicKey().getEncoded())+"\n");
                    msgShowArea.append("Client RSA public
key:"+byteToBinary(clientPBK.getEncoded())+"\n");

msgShowArea.setCaretPosition(msgShowArea.getText().length());
                }
            }
            catch(Exception e)
            {
              System.out.println("keyThread: "+e);
            }
        }
    }

    class keyThreadDH extends Thread
    {
        private ObjectOutputStream ObjOSDES;
        private ObjectOutputStream ObjOSDH;
        private ObjectInputStream ObjISDH;
        byte[] keyDESbyte;

        // Δημιουργία των streams στα αντίστοιχα sockets για τον DH
        public keyThreadDH(Socket sDH, Socket sDES) throws IOException
        {
            ObjOSDES = new ObjectOutputStream(sDES.getOutputStream());
            ObjOSDH = new ObjectOutputStream(sDH.getOutputStream());
            ObjISDH = new ObjectInputStream(sDH.getInputStream());
        }

        public void run()
        {
            try
            {
               /* Ο server έχει λάβει το δημόσιο κλειδί του client σε
                κωδικοποιημένη μορφή. Με το υλικό του κλειδιού αρχικοποιεί
                ένα DH δημόσιο κλειδί.*/
              byte[] clientPubKeyEncDH = (byte[]) ObjISDH.readObject();

       KeyFactory serverKeyFac = KeyFactory.getInstance("DH");
         X509EncodedKeySpec x509KeySpec =
                            new X509EncodedKeySpec(clientPubKeyEncDH);
     PublicKey clientPubKey = serverKeyFac.generatePublic(x509KeySpec);

      /*Ο server λαμβάνει τις παραμέτρους DH που συνδέονται με το
        δημόσιο κλειδί του client. Θα πρέπει να χρησιμοποιήσει τις
        ίδιες παραμέτρους όταν παράγει το δικό του ζεύγος κλειδιών DH.*/
DHParameterSpec dhParamSpec = ((DHPublicKey) clientPubKey).getParams();
```

```
// Ο server παράγει το δικό του ζεύγος κλειδιών DH.
KeyPairGenerator serverKpairGen = KeyPairGenerator.getInstance("DH");
                serverKpairGen.initialize(dhParamSpec);
                KeyPair serverKpair = serverKpairGen.generateKeyPair();

// Ο server δημιουργεί και αρχικοποιεί το δικό του αντικείμενο KeyAgreement.
          KeyAgreement serverKeyAgree = KeyAgreement.getInstance("DH");
          serverKeyAgree.init(serverKpair.getPrivate());

// Ο server κωδικοποιεί το δημόσιο κλειδί του και το στέλνει στον client
     byte[] serverPubKeyEncDH = serverKpair.getPublic().getEncoded();
                 ObjOSDH.writeObject(serverPubKeyEncDH);
                 ObjOSDH.flush();
// Η μέθοδος doPhase στο server εκτελεί την επόμενη φάση της συμφωνίας
// κλειδιού με το δοθέν κλειδί (clientPubKey) που έλαβε από τον client
                 serverKeyAgree.doPhase(clientPubKey, true);

                 int clientLen = (Integer) ObjISDH.readObject();
                 byte[] serverSharedSecret = new byte[clientLen];

                 ObjOSDH.close();
                ObjISDH.close();

// Η generateSecret παράγει το δημόσιο κλειδί (αυτό που γίνεται share) και το
// τοποθετεί στο serverSharedSecret
                 serverKeyAgree.generateSecret(serverSharedSecret, 0);

System.out.println("Server secret (Hex format): " +
toHexString(serverSharedSecret)+"\nServer secret (Binary
format):"+byteToBinary(serverSharedSecret));

// Παράγεται το μυστικό κλειδί του DES που ανταλλάχθηκε μέσω του πρωτοκόλλου
// DH και είναι αυτό που θα χρησιμοποιηθεί για την αποκρυπτογράφηση του
// μηνύματος
serverDesKeyDH = serverKeyAgree.generateSecret("DES");

                msgShowArea.append("Server DH private
key:"+byteToBinary(serverSharedSecret)+"\n");
                msgShowArea.append("Server DH public
key:"+byteToBinary(serverPubKeyEncDH)+"\n");
                msgShowArea.append("Client DH public
key:"+byteToBinary(clientPubKeyEncDH)+"\n");
                msgShowArea.append("DES secret
key:"+byteToBinary(serverDesKeyDH.getEncoded()));
                msgShowArea.setCaretPosition(msgShowArea.getText().length());
            }
            catch(Exception e)
            {
             System.out.println("keyThread: "+e);
            }
        }
    }

    class recThread extends Thread
    {
        private byte[] eData;
        private byte[] data;
```

```
        private String str;

        public recThread(Socket c) throws IOException
        {
            ObjOSMsg= new ObjectOutputStream(c.getOutputStream());
            ObjISMsg= new ObjectInputStream(c.getInputStream());
        }

        public void run()
        {
            try
            {
                while(true)
                {
                    eData = (byte[]) ObjISMsg.readObject();

                    // Αποκρυπτογράφηση
                    Long startMilliTime = System.currentTimeMillis();
                 if(algorithm.equalsIgnoreCase("des"))
                      data = DES.SEnc(keyDES, "DEC", eData);
                 else if (algorithm.equalsIgnoreCase("rsa"))

                     data = serverKeyRSA.decryptRSA
                                (serverKeyRSA.getPrivateKey(), eData);
                    else if (algorithm.equalsIgnoreCase("dh"))
                    {
                    data = DES.SEnc(serverDesKeyDH, "DEC", eData);
                    }
                    Long finishMilliTime = System.currentTimeMillis();

                    str = new String(data);

                    msgShowArea.append("\n**Client: "+str+"\n");
                    msgShowArea.append("\n-Η αποκρυπτογράφηση
("+algorithm.toUpperCase()+") ξεκίνησε στις: "+startMilliTime);
                    msgShowArea.append("\n-Η αποκρυπτογράφηση
("+algorithm.toUpperCase()+") έληξε στις: "+finishMilliTime);
                    msgShowArea.append("\n-Διάρκεια: "+(finishMilliTime-
startMilliTime)+" milliseconds.");
                    msgShowArea.append("\n-Μέγεθος καθαρού μηνύματος:
"+data.length+" χαρακτήρες.");
                    msgShowArea.append("\n-Μέγεθος κρυπτογραφημένου
μηνύματος: "+eData.length+" χαρακτήρες.");

msgShowArea.setCaretPosition(msgShowArea.getText().length());
                }
            }
            catch(Exception e)
            {
              System.out.println(e);
            }
        }
    }

    private void initMsgShowPanel()
    {
        msgShowPanel = new JPanel();
```

```
        msgShowPanel.setLayout(new BorderLayout());

        JLabel label = new JLabel("Ιστορικό Μηνυμάτων: ");
        label.setFont(font);

        msgShowArea = new JTextArea(20,65);
        msgShowArea.setEditable(false);
        JScrollPane msgShowPane = new JScrollPane();
        msgShowPane.setViewportView(msgShowArea);

        msgShowPanel.add(label,BorderLayout.NORTH);
        msgShowPanel.add(msgShowPane,BorderLayout.CENTER);
    }

    // Επιστρέφει σε δυαδική μορφή (τύπου String) το περιεχόμενο του buffer.
    private static String byteToBinary (byte[] bytes)
    {
        BigInteger bi = new BigInteger(bytes);
        return bi.toString(2);
    }

// Επιστρέφει σε δεκαεξαδική μορφή (τύπου String) το περιεχόμενο του buffer.
    private void byte2hex(byte b, StringBuffer buf)
    {
    char[] hexChars = { '0', '1', '2', '3', '4', '5', '6', '7', '8',
    '9', 'A', 'B', 'C', 'D', 'E', 'F' };
    int high = ((b & 0xf0) >> 4);
    int low = (b & 0x0f);
    buf.append(hexChars[high]);
    buf.append(hexChars[low]);
    }

    // Μετατρέπει το μήνυμα από δεκαεξαδική μορφή σε String.
    private String toHexString(byte[] block)
    {
   StringBuffer buf = new StringBuffer();

   int len = block.length;

      for (int i = 0; i < len; i++)
      {
          byte2hex(block[i], buf);
          if (i < len - 1)
          {
              buf.append(":");
          }
      }
      return buf.toString();
    }
}

// Κώδικας του αρχείου DES.java
import javax.crypto.*;
/* Η κλάση DES έχει δύο πεδία που αποτελούν τις παραμέτρους που απαιτούνται
   στη διαδικασία παραγωγής και χρήσης των κλειδιών κρυπτογράφησης /
   αποκρυπτογράφησης.*/
public class DES
```

```
{
    private SecretKey sKey;
    private static Cipher cp;

    public DES() throws Exception
    {
        KeyGenerator kg = KeyGenerator.getInstance("DES");
        //JCE Keysize Restriction - 56 bit
        kg.init(56);
        // Παραγωγή του μυστικού κλειδιού
        sKey = kg.generateKey();
    }
    // Κρυπτογράφηση/αποκρυπτογράφηση μηνύματος μέσω DES.
    // Μέγεθος κλειδιού 56 bit.
    public static byte[] SEnc(SecretKey k, String mode, byte[] data)
            throws Exception
    {
        cp = Cipher.getInstance("DES");
        if(mode.equalsIgnoreCase("dec"))
            cp.init(Cipher.DECRYPT_MODE, k);
        else
            cp.init(Cipher.ENCRYPT_MODE, k);
// Ανάλογα με την επιλογή dec ή enc έχουμε την κρυπτογράφηση ή την
// αποκρυπτογράφηση του data
        return cp.doFinal(data);
    }

    public SecretKey getSecretKey()
    {
        return sKey;
    }
}

// Κώδικας του αρχείου RSA.java
import java.io.UnsupportedEncodingException;
import java.security.*;

import javax.crypto.*;
import org.bouncycastle.util.encoders.Hex;

public class RSA
{
    private PublicKey pbkey;
    private PrivateKey prkey;
    private  Cipher cipher;

    // Δημιουργία του ζεύγους κλειδιών RSA
    public RSA() throws Exception
    {
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(1024);
        KeyPair kp = kpg.genKeyPair();
        pbkey = kp.getPublic();
        prkey = kp.getPrivate();

        cipher = Cipher.getInstance("RSA");
    }
```

```
    // Κρυπτογράφηση δεδομένων τύπου String με το δημόσιο κλειδί του RSA
    public byte[] encryptRSA(PublicKey publicKey, String data) throws
NoSuchAlgorithmException, NoSuchPaddingException,
UnsupportedEncodingException, InvalidKeyException, IllegalBlockSizeException,
BadPaddingException
    {
        this.cipher.init(Cipher.ENCRYPT_MODE, publicKey);

        byte[] bytes = data.getBytes("UTF-8");
        byte[] encrypted = blockCipher(bytes,Cipher.ENCRYPT_MODE);
        byte[] encryptedTranspherable = Hex.encode(encrypted);
        return encryptedTranspherable;
    }

// Κρυπτογράφηση δεδομένων τύπου byte (buffer) με το δημόσιο κλειδί του RSA
    public byte[] encryptRSA(PublicKey publicKey, byte[] data) throws
NoSuchAlgorithmException, NoSuchPaddingException,
UnsupportedEncodingException, InvalidKeyException, IllegalBlockSizeException,
BadPaddingException
    {
        this.cipher.init(Cipher.ENCRYPT_MODE, publicKey);

        byte[] encrypted = blockCipher(data,Cipher.ENCRYPT_MODE);
        byte[] encryptedTranspherable = Hex.encode(encrypted);
        return encryptedTranspherable;
    }

// Αποκρυπτογράφηση δεδομένων τύπου byte (buffer) με το ιδιωτικό κλειδί του
// RSA
    public byte[] decryptRSA(PrivateKey privateKey, byte[] encrypted) throws
NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
IllegalBlockSizeException, BadPaddingException
    {
        this.cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] bts = Hex.decode(encrypted);

        byte[] decrypted = blockCipher(bts,Cipher.DECRYPT_MODE);

        return decrypted;
     }

// Επειδή ο RSA αποτελεί block cipher απαιτείται αυτή η βοηθητική μέθοδος //
για την κρυπτογράφηση μεγάλου μηνύματος.
    private byte[] blockCipher(byte[] bytes, int mode) throws
IllegalBlockSizeException, BadPaddingException
    {
      // Το buffer scrambled θα κρατά ενδιάμεσα αποτελέσματα
      byte[] scrambled = new byte[0];

      // Το buffer toReturn θα κρατά το τελικό αποτέλεσμα
      byte[] toReturn = new byte[0];
      // Αν κρυπτογραφούμε θα χρησιμοποιούμε block των 100 byte.
            // Εξαιτίας του RSA η αποκρυπτογράφηση απαιτεί block των 128
            // byte.
      int length = (mode == Cipher.ENCRYPT_MODE)? 100 : 128;
      byte[] buffer = new byte[length];
```

```
        for (int i=0; i< bytes.length; i++)
        {
            if ((i > 0) && (i % length == 0))
            {
                scrambled = cipher.doFinal(buffer);
                toReturn = append(toReturn,scrambled);
                int newlength = length;

// Αν το νέο μήκος είναι μεγαλύτερο από τα εναπομύναντα bytes στον πίνακα //
bytes τον κάνουμε μικρότερο.
                if (i + length > bytes.length)
                {
                     newlength = bytes.length - i;
                }
                // Καθαρίζουμε το buffer
                buffer = new byte[newlength];
            }
            buffer[i%length] = bytes[i];
        }

        scrambled = cipher.doFinal(buffer);
        toReturn = append(toReturn,scrambled);

        return toReturn;
    }

     // Βοηθητική μέθοδος της blockCipher
     private byte[] append(byte[] prefix, byte[] suffix)
     {
        byte[] toReturn = new byte[prefix.length + suffix.length];
            for (int i=0; i< prefix.length; i++)
            {
                toReturn[i] = prefix[i];
            }
            for (int i=0; i< suffix.length; i++)
            {
                toReturn[i+prefix.length] = suffix[i];
            }
            return toReturn;
        }

    // Χρησιμοποιείται όταν θέλουμε να μεταδώσουμε το μυστικό κλειδί
        // συμμετρικού αλγορίθμου (DES) μέσω του RSA
     public static byte [] wrapkey(Key key, PublicKey publicKey)
         throws Exception
     {
         Cipher cipher = Cipher.getInstance("RSA");
         cipher.init(Cipher.WRAP_MODE, publicKey);
         return (cipher.wrap(key));
     }

     // Χρησιμοποιείται όταν θέλουμε να λάβουμε το μυστικό κλειδί
     // συμμετρικού αλγορίθμου (DES) μέσω του RSA
     public static Key unwrapkey(
         byte []wrapedkey,
         PrivateKey privateKey,
```

```
        String wrappedKeyAlgorithm,
        int wrappedKeyType) throws Exception
    {

         Cipher cipher = Cipher.getInstance("RSA");
         cipher.init(Cipher.UNWRAP_MODE, privateKey);
         return (cipher.unwrap(wrapedkey, wrappedKeyAlgorithm,
wrappedKeyType));
     }

    public PublicKey getPublicKey()
    {
        return pbkey;
    }

    public PrivateKey getPrivateKey()
    {
        return prkey;
    }
}

// Κώδικας του αρχείου DH.java
import java.security.AlgorithmParameterGenerator;
import java.security.AlgorithmParameters;

import javax.crypto.spec.DHParameterSpec;

public class DH
{
    private DHParameterSpec dhSkipParamSpec;
    private AlgorithmParameterGenerator paramGen;
    private AlgorithmParameters params;

    // Δημιουργία των παραμέτρων Dh
    public DH() throws Exception
    {
        paramGen = AlgorithmParameterGenerator.getInstance("DH");
        //512 bit OR 1024 bit
        paramGen.init(512);
        params = paramGen.generateParameters();
        dhSkipParamSpec = (DHParameterSpec)
       params.getParameterSpec(DHParameterSpec.class);
    }

    public DHParameterSpec getDhSkipParamSpec() {
       return dhSkipParamSpec;
    }
}
```