ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Μελέτη πρωτοκόλλου TESLA για ασφαλή μετάδοση δεδομένων σε συστήματα πολυεκπομπής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΜΥΡΩΝ Χ. ΑΠΟΣΤΟΛΑΚΗ

Επιβλέπων : Παναγιώτης Φράγκος
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2007

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Μελέτη πρωτοκόλλου *TESLA* για ασφαλή μετάδοση δεδομένων σε συστήματα πολυεκπομπής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΜΥΡΩΝ Χ. ΑΠΟΣΤΟΛΑΚΗ

Επιβλέπων : Παναγιώτης Φράγκος
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Ιουνίου 2007.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Παναγιώτης Φράγκος	Μιχαήλ Θεολόγου	Γεώργιος Κολέτσος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2007

(Υπογραφή)

.....

ΜΥΡΩΝ Χ. ΑΠΟΣΤΟΛΑΚΗΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Πολλές προκλήσεις αναδύονται από την χρήση των τεχνολογιών πληροφορικής και επικοινωνιών (Information Communication Technology ή συντομογραφικά ICT). Μια σημαντική κατηγορία αυτών εντοπίζονται στο χώρο της μετάδοσης της πληροφορίας και στα συστήματα εκπομπής δεδομένων. Ένας μέσος χρήστης συστημάτων εκπομπής και λήψης δεδομένων, για παράδειγμα της τηλεόρασης ή του ραδιοφώνου, μπορεί να παρατηρήσει εύκολα αρκετές από αυτές. Πιο συγκεκριμένα οι προκλήσεις είναι τεσσάρων ειδών. Αρχικά ο δέκτης - χρήστης του συστήματος θα πρέπει να εξασφαλίσει την αυθεντικότητα της πηγής της εκπομπής των δεδομένων, δηλαδή να αποκτήσει τη γνώση της ταυτότητας του αποστολέα και να μπορεί να αποφασίσει αν τα δεδομένα προέρχονται πράγματι από τον ισχυριζόμενο αποστολέα. Κατόπιν ένα σύστημα εκπομπής δεδομένων θα πρέπει να αντεπεξέρχεται επιτυχώς σε μεγάλους αριθμούς δεκτών. Ένα τρίτο είδος προκλήσεων είναι η απώλεια των πακέτων δεδομένων κατά την μετάδοση. Τέλος το σύστημα θα πρέπει να πετυχαίνει υψηλούς ρυθμούς μετάδοσης των πληροφοριών. Όλες αυτές οι προκλήσεις συναντώνται σε ποικίλα συστήματα εκπομπής δεδομένων όπως για παράδειγμα το διαδίκτυο ή η μετάδοση δεδομένων μέσω δορυφόρου.

Σκοπός της παρούσας διπλωματικής εργασίας είναι η μελέτη του πρωτοκόλλου αυθεντικοποίησης *TESLA*. Το *TESLA* είναι ένα αποδοτικό πρωτόκολλο με μικρές υπολογιστικές απαιτήσεις και μικρό φόρτο επικοινωνίας, το οποίο βασίζεται στον μη αυστηρό συγχρονισμό μεταξύ του αποστολέα και των παραληπτών. Ακόμη το *TESLA* μπορεί και αντεπεξέρχεται σε μεγάλο αριθμό παραληπτών, είναι ανθεκτικό σε τυχαίο αριθμό απωλειών των πακέτων ενώ πετυχαίνει και υψηλούς ρυθμούς μετάδοσης. Τέλος παρουσιάζεται και μια υλοποίηση του *TESLA* στη γλώσσα JAVA.

Λέξεις Κλειδιά:

Ασφάλεια, αυθεντικοποίηση, συμμετρική και ασύμμετρη κρυπτογραφία, ψευδοτυχαίες συναρτήσεις, μονόδρομες αλυσίδες, συναρτήσεις κατακερματισμού, ψηφιακές υπογραφές, συστήματα πολυεκπομπής δεδομένων, TESLA, JAVA

Abstract

There is a wide range of new challenges that arise from the use of Information Communicatoin Technologies (ICT). Some of them lie in the fields of information transmition and data transmission systems. The average user of a transmition and receival data system, for example that of a television or a radio system, will be able to identify several of them. These challenges can be categorized in four basic categories. Initially the receiver-user will have to certify the authenticity of the data source, through retreiving the sender identity and then to decide if the incoming data come indeed from the alleged sender. With respect to the second challenge, a data transmition and receival system will have to fulfill succesfully, is to accept and serve a large number of clients. A third challenge is to serve all these clients while at the same time maintaining the minimum data packet loss during their transmittion. Regarding the fourth and final challenge, the system has to face is to achieve the required high data transmittion rates, which when combined with the other three challenges, result in a high availability and high quality data transmition system. All the above mentioned challenges can be met in a variety of data transmition systems, for example a wide variety of internet protocols, like WWW, VoIP, banking applications, digital television as well as other transmition protocols like satelite data transmition etc.

The aim of this project is to study the authentication protocol named *TESLA*. *TESLA* is an efficient protocol with small computational and communication overhead, based on the loosely synchronization between the sender and the receivers. It can accept a large number of receivers, can resolve succesfully a random number of data packet loss, while achieving high data transmition rates. Finally an implementation of the *TESLA* procotol is presented as part of this project.

Keywords:

Security, authentication, symmetric and asymmetry cryptography, pseudorandom functions, one way chains, hash functions, digital signatures, multicast data systems, TESLA, JAVA

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω θερμά την οικογένεια μου για τη αμέριστη στήριξη, συμπαράσταση και εμπιστοσύνη που έδειξαν στο πρόσωπο μου κατά τη διάρκεια των σπουδών μου. Ακόμη ευχαριστώ ιδιαίτερα τον επιβλέποντα καθηγητή, Παναγιώτη Φράγκο και τον υποψήφιο διδάκτορα Ουρανό Ιάκωβο για την καθοδήγηση και τις πολύτιμες συμβουλές τους που μου παρείχαν κατά την εκπόνηση της παρούσας διπλωματικής εργασίας. Τέλος, θερμές ευχαριστίες στον φίλο και συνάδελφο Ορφανουδάκη Γεώργιο για τις εποικοδομητικές συζητήσεις και την άριστη συνεργασία που είχαμε όλα αυτά τα χρόνια.

-Αφιερώνεται στην οικογένεια μου-

Πίνακας περιεχομένων

1

Εισαγωγή

1.1 Γενικά

Πολλά προβλήματα αναδύονται από την χρήση της τεχνολογίας και των υπολογιστών, η οποία είναι πολλές φορές αυτοεπιβαλόμενη στον σύγχρονο άνθρωπο. Μια σημαντική κατηγορία αυτών εντοπίζεται στον χώρο της μετάδοσης της πληροφορίας και στα συστήματα εκπομπής των δεδομένων. Δεν χρειάζεται να κατέχει κάποιος ιδιαίτερες γνώσεις για να αντιληφθεί την ύπαρξη αυτών των προβλημάτων τα οποία γίνονται πολλές φορές αντιληπτά δια γυμνού οφθαλμού. Ένα παράδειγμα αποτελεί η απώλεια σκηνών κατά τη διάρκεια μιας ταινίας ή ακόμη και η διακοπή του ήχου για μερικά δευτερόλεπτα κατά τη διάρκεια ενός τραγουδιού. Πιο συγκεκριμένα τα προβλήματα που προκύπτουν, κατατάσσονται σε τέσσερα είδη. Η πρώτη κατηγορία αφορά την αυθεντικότητα του αποστολέα των δεδομένων. Δηλαδή κατά πόσο ο χρήστης-δέκτης του συστήματος μπορεί να αποκτήσει τη γνώση της ταυτότητας του αποστολέα και να αποφασίσει αν πράγματι τα δεδομένα προέρχονται από τον ισχυριζόμενο αποστολέα. Η διαδικασία αυτή λέγεται αυθεντικοποίηση των δεδομένων και αποτελεί ένα από τους σημαντικότερους τομείς έρευνας της πληροφορικής. Ένα παράδειγμα των προβλημάτων αυτού του είδους είναι η παράνομη εισαγωγή χρηστών (*hackers*) σε συστήματα βάσεων δεδομένων διάφορων υπηρεσιών (π.χ. σε κρατικές ή στρατιωτικές υπηρεσίες) και η υποκλοπή ή η διαστρέβλωση πληροφοριών από αυτά. Η δεύτερη κατηγορία αναφέρεται στον αριθμό των χρηστών - δεκτών του συστήματος. Ένα αξιόπιστο και αποδοτικό σύστημα θα πρέπει να ανταποκρίνεται επιτυχώς ακόμη και όταν υπάρχουν συγχρόνως μεγάλα νούμερα αποδεκτών. Ένα παράδειγμα των προβλημάτων της κατηγορίας αυτής παρατηρείται κατά τη διάρκεια των γιορτών και ειδικά την παραμονή της πρωτοχρονιάς, όπου ανταλλάσσονται πολλά ευχητήρια *SMS* με αποτέλεσμα την υπερφόρτωση των τηλεπικοινωνιακών συστημάτων των εταιριών κινητών τηλεφωνίας. Η τρίτη κατηγορία συμπεριλαμβάνει τα προβλήματα που προκύπτουν από πιθανή απώλεια των πακέτων δεδομένων αποστολής. Παραδείγματα αυτής της κατηγορίας αποτελούν τα αρχικά προαναφερθέντα. Τέλος το σύστημα θα πρέπει να υποστηρίζει υψηλούς ρυθμούς μετάδοσης των πακέτων. Προβλήματα αυτού του είδους συναντώνται συχνά κατά την χρήση του διαδικτύου όπου δεν υπάρχει γρήγορη λήψη δεδομένων.

1.2 Αντικείμενο διπλωματικής

Κάθε κατηγορία προβλημάτων που παρουσιάστηκε στη προηγούμενη ενότητα αποτελεί πρόκληση και αντικείμενο έρευνας. Η παρούσα διπλωματική παρουσιάζει το πρωτόκολλο αυθεντικοποίησης *TESLA* και μια υλοποίηση αυτού στη γλώσσα *JAVA*. Το *TESLA* είναι ένα αξιόπιστο και αποδοτικό πρωτόκολλο με μικρές υπολογιστικές απαιτήσεις. Η βασική ιδέα του *TESLA* έχει ως εξής: Το *TESLA* χρησιμοποιεί μόνο συμμετρικές κρυπτογραφικές θεμελιώδεις λειτουργίες όπως οι ψευδοτυχαίες συναρτήσεις *(PRFs)* και τους κώδικες αυθεντικότητας μηνυμάτων *(MACs ή message authentication codes)* ενώ παράλληλα βασίζεται στη μη αυστηρά συγχρονισμένη αποκάλυψη των κλειδιών από τον αποστολέα. Πιο συγκεκριμένα ο αποστολέας δεσμεύεται σε ένα τυχαίο κλειδί *k* το οποίο το κρατάει κρυφό. Μεταδίδει στους παραλήπτες τη δέσμευση στο κλειδί. Η δέσμευση πραγματοποιείται μέσω μιας ψευδοτυχαίας συνάρτησης (PRF). Ο αποστολέας στη συνέχεια προσαρτεί έναν κωδικό αυθεντικότητας μηνύματος στο επόμενο πακέτο P_i και χρησιμοποιεί το κλειδί *k* ως το κλειδί για να κατασκευάσει τον *MAC*. Παράλληλα όλα τα κλειδιά που χρησιμοποιούνται για τις *MACs* σχηματίζουν μια μονόδρομη αλυσίδα *(one-way key chain)*. Σε ένα μεταγενέστερο πακέτο P_{i+1} ο αποστολέας αποκαλύπτει το *k* επομένως οι παραλήπτες μπορούν να κατασκευάσουν την *MAC* του P_i . Εάν και οι δύο επικυρώσεις είναι σωστές και είναι εγγυημένο ότι το P_{i+1} δεν εστάλη πριν το P_i τότε ο παραλήπτης γνωρίζει σίγουρα ότι το πακέτο είναι αυθεντικό. Για την αρχικοποίηση του πρωτοκόλλου ο αποστολέας χρησιμοποιεί ένα συνηθισμένο αλγόριθμο υπογραφής για να υπογράψει το αρχικό πακέτο που στέλνει σε κάθε παραλήπτη. Το αρχικό πακέτο περιέχει το κλειδί στο οποίο δεσμεύτηκε ο αποστολέας. Όλα τα πακέτα αυθεντικοποιούνται μέσω των αλυσίδων που σχηματίζουν τα κλειδιά.

1.3 Οργάνωση κειμένου

Η διπλωματική οργανώνεται στα παρακάτω κεφάλαια:

1) Στο 1^{ο} κεφάλαιο γίνεται μία εισαγωγή στο πρόβλημα με το οποίο ασχολείται η εργασία και δίνεται συνοπτικά η δομή της εργασίας.
2) Στο 2^{ο} κεφάλαιο αναπτύσσονται κάποιες βασικές γνώσεις που αφορούν τους τομείς με τους οποίους σχετίζεται η εργασία (ασφάλεια, κρυπτογραφία, συστήματα εκπομπής δεδομένων).
3) Στο 3^{ο} κεφάλαιο παρουσιάζεται το πρωτόκολλο αυθεντικοποίησης *TESLA* σε λεπτομέρεια.
4) Στο 4^{ο} κεφάλαιο παρουσιάζεται η ανάλυση απαιτήσεων και η αρχιτεκτονική του συστήματος. Αρχικά παρουσιάζονται τα ανεξάρτητα υποσυστήματα που το αποτελούν, ενώ στη συνέχεια οι εφαρμογές που υλοποιεί το σύστημα.

5) Στο 5^{ο} κεφάλαιο παρουσιάζεται η σχεδίαση του συστήματος.

6) Στο 6^{ο} κεφάλαιο παρουσιάζεται η υλοποίηση του συστήματος και αναφέρονται εν συντομία οι πλατφόρμες και τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν.

7) Στο 7^{ο} κεφάλαιο δίνονται σενάρια ελέγχου του συστήματος με το οποίο γίνεται σύγκριση των αποτελεσμάτων με τα αναμενόμενα.

8) Στο 8^{ο} κεφάλαιο επιχειρείται μία σύνοψη της εργασίας και παρουσιάζονται πιθανές μελλοντικές επεκτάσεις και θέματα γύρω από το αντικείμενο αυτής της διπλωματικής εργασίας.

9) Στο 9^{ο} κεφάλαιο παρουσιάζεται η σχετική βιβλιογραφία.

2

Θεωρητικό υπόβαθρο

2.1 Βασικές αρχές ασφάλειας υπολογιστικών συστημάτων

2.1.1 Εισαγωγή

Η ασφάλεια των υπολογιστικών συστημάτων είναι ένα πεδίο της επιστήμης της πληροφορικής, το οποίο συνδέεται με τον έλεγχο των κινδύνων που αφορούν την χρήση των υπολογιστών. Συνήθως για να αντιληφτεί κανείς πλήρως το αντικείμενο της επιστήμης της ασφάλειας των υπολογιστικών συστημάτων θα πρέπει να προσπαθήσει να δημιουργήσει μια έμπιστη και ασφαλή πλατφόρμα, σχεδιασμένη κατά τέτοιον τρόπο ώστε οι οντότητες που την χρησιμοποιούν να μπορούν να εκτελέσουν μόνο επιτρεπτές ενέργειες. Αυτό προϋποθέτει τον ορισμό και την υλοποίηση μιας πολιτικής η οποία εγγυάται την ασφάλεια του συστήματος *(security policy)*. Η ασφάλεια των υπολογιστικών συστημάτων μπορεί να θεωρηθεί ως ένα υποπεδίο της μηχανικής ασφάλειας *(security engineering),* η οποία ερευνά ευρύτερα θέματα ασφάλειας.

Σε ένα ασφαλή σύστημα οι εξουσιοδοτημένοι χρήστες διατηρούν τα προνόμια που θα είχαν εξαρχής. Σύμφωνα με την άποψη του Eugene H. Spafford, διευθυντή του κέντρου, για την εκπαίδευση και έρευνα στην ασφάλεια της πληροφορίας, Purdue

> “Το μοναδικό αληθινά ασφαλές σύστημα είναι το σβηστό, το οποίο περιορίζεται σε ένα συμπαγή, σφραγισμένο και φρουρούμενο δωμάτιο. Όμως ακόμα και τότε θα υπήρχαν αμφιβολίες.”

Το μειονέκτημα σ’ αυτήν την περίπτωση, είναι ότι το εν λόγω σύστημα δεν θεωρείται ως ένα χρήσιμο ασφαλές σύστημα. Γι ’αυτό είναι σημαντικό να διαχωρίσουμε τις τεχνικές που χρησιμοποιούνται για να αυξήσουμε την ασφάλεια ενός συστήματος, από την ακραία περίπτωση ασφάλειας εκείνου του συστήματος. Πιο συγκεκριμένα συστήματα που περιέχουν μεγάλες ελλείψεις στη σχεδίαση της ασφάλειάς τους δεν μπορούν να γίνουν ασφαλή, χωρίς να μειώσουν τις δυνατότητες των χρήσεων τους. Συνεπώς, τα περισσότερα συγκροτήματα

ηλεκτρονικών υπολογιστών δεν μπορούν να γίνουν ασφαλή ακόμα και μετά από την εφαρμογή των εκτενών μέτρων «ασφάλειας υπολογιστών». Επιπλέον, εάν γίνονται ασφαλείς, συχνά είναι εις βάρος της δυνατότητας χρησιμοποίησης τους.

Οι τεχνικές που χρησιμοποιήθηκαν στην παρούσα εργασία είναι οι τεχνικές της κρυπτογραφίας, των ψηφιακών υπογραφών, των μονόδρομων αλυσίδων και συναρτήσεων οι οποίες και θα παρουσιαστούν στη συνέχεια.

2.1.2 Κρυπτογραφία

Κρυπτογραφία ή κρυπτολογία είναι η μελέτη της μυστικότητας των μηνυμάτων (*secrecy of messages*). Στις μέρες μας, έχει γίνει κλάδος της θεωρίας της πληροφορίας *(information theory)* σαν η μαθηματική μελέτη της πληροφορίας και πιο συγκεκριμένα η μελέτη κατά την μετάδοση αυτής. Ο διακεκριμένος κρυπτογράφος Ron Rivest έχει παρατηρήσει ότι "κρυπτογραφία είναι η επικοινωνία με την παρουσία εισβολέων". Έχει συνεισφέρει σε διάφορα πεδία της επιστήμης όπως η ασφάλεια της πληροφορίας *(information security)*, στην αυθεντικοποίηση ταυτότητας *(authentication)* καθώς και στον έλεγχο πρόσβασης σε διάφορες υπηρεσίες ενός συστήματος.

Ένας από τους κύριους στόχους της κρυπτογραφίας είναι η απόκρυψη της σημασίας των μηνυμάτων και όχι την απόκρυψη της ύπαρξης αυτών. Επίσης η κρυπτογραφία έχει συνεισφέρει στην πληροφορική και περισσότερο στην ασφάλεια των δικτύων υπολογιστών στους τομείς του ελέγχου πρόσβασης χρηστών *(access control)* και στην εμπιστευτικότητα της πληροφορίας *(information confidentiality)*. Η κρυπτογραφία χρησιμοποιείται ακόμη σε πολλές εφαρμογές οι οποίες συναντούνται συχνά στην καθημερινότητα του σύγχρονου ανθρώπου, για παράδειγμα στην ασφάλεια των καρτών ATM, στα *password* που ζητούνται κατά την χρήση των υπολογιστών καθώς και στο ηλεκτρονικό εμπόριο *(electronic commerce)*.

2.1.2.1 Ορολογία

Ο όρος αναφέρεται συχνά στο σύνολο του πεδίου όπως και η κρυπτολογία (“ η μελέτη των μηνυμάτων ”). Η μελέτη του πώς να ξεγελάσει κανείς την εμπιστευτικότητα των μηνυμάτων χρησιμοποιώντας την απόκρυψη τους, καλείται κρυπτανάλυση *(cryptanalysis)*. Μέχρι πρότινος η κρυπτογραφία αφορούσε σχεδόν αποκλειστικά την *απόκρυψη* των μηνυμάτων, δηλαδή την διαδικασία της μετατροπής συνηθισμένων πληροφοριών, για παράδειγμα απλού κειμένου, σε μια ακατανόητη μορφή πληροφορίας. Αυτή η ακατανόητη μορφή πληροφορίας λέγεται κρυπτογραφημένο κείμενο *(ciphertext)* ή κρυπτοκείμενο. Η αποκρυπτογράφηση είναι η αντίθετη διαδικασία κατά την οποία ένα κρυπτογραφημένο κείμενο μετατρέπεται σε μια

πλήρως κατανοητή μορφή πληροφορίας, για παράδειγμα σε απλό κείμενο. Το ζευγάρι των αλγορίθμων που εφαρμόζεται κατά την κωδικοποίηση και την αποκωδικοποίηση της πληροφορίας ονομάζεται κρυπτογράφημα *(cipher)*. Οι λεπτομερείς ενέργειες που λαμβάνουν χώρα ελέγχονται και στις δύο περιπτώσεις από ένα κλειδί *(key)*. Αυτό είναι μια μυστική παράμετρος, η οποία είναι γνωστή μόνο στους επικοινωνούντες. Τα κλειδιά πρέπει να μεταβάλλονται κατά τυχαίο τρόπο αφού σε αντίθετη περίπτωση κάποιος εισβολέας θα μπορεί να τα υποκλέψει εύκολα και να αλλοιώσει έτσι τα μηνύματα που μεταδίδονται.

Στην καθομιλουμένη, ο όρος κωδικοποιώ *(code)* χρησιμοποιείται συχνά για να δηλώσει μια μέθοδο κρυπτογράφησης ή απόκρυψης της σημασίας κάποιων μηνυμάτων. Όμως στην κρυπτογραφία σημαίνει την αντικατάσταση μιας μονάδας απλού κειμένου (π.χ. μιας λέξης ή φράσης) με μια κωδικοποιημένη λέξη (για παράδειγμα, η λέξη "μηλόπιτα" θα μπορούσε να αντικαταστήσει τη φράση "επίθεση την αυγή") . Οι κωδικοποιημένες λέξεις και φράσεις δεν χρησιμοποιούνται πλέον στην κρυπτογραφία αφού κατάλληλα επιλεγμένα ciphers είναι πολύ πιο πρακτικά, πιο ασφαλή και πιο υλοποιήσιμα σε υπολογιστή από τις καλύτερες κωδικοποιημένες λέξεις και φράσεις. Τέλος μερικοί χρησιμοποιούν τους όρους κρυπτογραφία και κρυπτανάλυση χωρίς διάκριση, ενώ άλλοι χρησιμοποιούν τον όρο κρυπτογραφία για να αναφερθούν στην χρήση και εφαρμογή των τεχνικών κρυπτογράφησης και τον όρο κρυπτολογία για να αναφερθούν στην μελέτη της θεωρίας του συγκεκριμένου πεδίου της επιστήμης. Εμείς εδώ υιοθετούμε την δεύτερη άποψη.

2.1.2.2 Σύγχρονη κρυπτογραφία

2.1.2.2.1 Κρυπτογραφία συμμετρικού κλειδιού

Η κρυπτογραφία του συμμετρικού κλειδιού αναφέρεται σε μεθόδους στις οποίες τόσο ο αποστολέας όσο και ο παραλήπτης των μηνυμάτων μοιράζονται από κοινού το ίδιο κλειδί (ή πιο σπάνια κλειδιά που είναι διαφορετικά, αλλά συσχετίζονται με κάποιο εύκολα υπολογίσιμο τρόπο). Αυτό ήταν το μόνο είδος απόκρυψης που ήταν δημόσια γνωστό μέχρι το 1976.

Η σύγχρονη μελέτη των κρυπτογραφημάτων συμμετρικού κλειδιού σχετίζεται κυρίως με την μελέτη των συμπαγών κρυπτογραφημάτων *(block ciphers)*, των κρυπτογραφημάτων ρεύματος δεδομένων *(stream ciphers)* και των εφαρμογών τους. Ένα συμπαγές κρυπτογράφημα λαμβάνει στην είσοδο του ένα απλό κείμενο συγκεκριμένου μεγέθους, ένα κλειδί και βγάζει στην έξοδο του ένα συμπαγές κρυπτογραφημένο κείμενο του ίδιου μεγέθους. Αφού τα μηνύματα συχνά υπερβαίνουν το επιτρεπόμενο μέγεθος των *blocks*, θα πρέπει να βρεθεί μια μέθοδος "πλεξίματος" των διαδοχικών *blocks*. Διάφορες μεθόδοι έχουν αναπτυχθεί, άλλες με καλύτερη απόδοση και άλλες με χειρότερη και θα πρέπει να ληφθούν

υπόψη με προσοχή, όταν χρησιμοποιείται ένα συμπαγές κρυπτογράφημα σε ένα κρυπτογραφικό σύστημα.

Τα *Data Encryption Standard (DES)* και *Advanced Encryption Standard (AES)* είναι συμπαγή κρυπτογραφήματα που έχουν καθοριστεί ως πρότυπα κρυπτογραφίας από την κυβέρνηση των Η.Π.Α. (παρ' όλα αυτά το πρότυπο *DES* τελικά αποσύρθηκε μετά την υιοθέτηση του *AES*). Παρά την κατάργηση του ως επίσημο πρότυπο, το *DES* παραμένει ακόμη αρκετά δημοφιλή και χρησιμοποιείται από μια μεγάλη κλίμακα εφαρμογών, όπως στα *ATM*, στα *e-mail* και στην ασφαλή απομακρυσμένη πρόσβαση *(secure remote access)*. Ακόμη έχουν προταθεί και σχεδιαστεί πολλά συμπαγή κρυπτογραφήματα ποικίλης ποιότητας στην απόδοση ασφάλειας από τα οποία κάποια έχουν κατά καιρούς ανατραπεί.

Τα κρυπτογραφήματα ρεύματος δεδομένων σε αντίθεση με τα συμπαγή, χρησιμοποιούν ως κλειδί ένα αυθαίρετου μεγέθους ρεύμα δεδομένων το οποίο το συνδυάζουν με το απλό κείμενο bit προς bit ή χαρακτήρα προς χαρακτήρα. Σε ένα κρυπτογράφημα ρεύματος δεδομένων, το ρεύμα εξόδου δημιουργείται βασιζόμενο στην εσωτερική κατάσταση του κρυπτογραφήματος η οποία μεταβάλλεται καθώς συνεχίζεται η διαδικασία. Αυτή η εσωτερική κατάσταση συνήθως ελέγχεται από το κλειδί και καμιά φορά και από το ρεύμα του κειμένου που κρυπτογραφείται. Το *RC4* είναι ένα παράδειγμα γνωστού κρυπτογραφήματος ρεύματος δεδομένων.

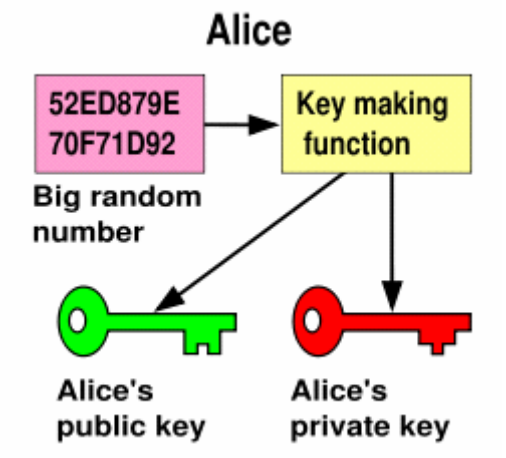
2.1.2.2.2 Κρυπτογραφία δημόσιου κλειδιού ή ασύμμετρη κρυπτογραφία (Public-key cryptography ή asymmetry cryptography)

Ένα σημαντικό μειονέκτημα των συστημάτων συμμετρικής κρυπτογραφίας είναι η διαχείριση των κλειδιών που μοιράζονται από κοινού ο αποστολέας με τον παραλήπτη έτσι ώστε να εξασφαλίζεται η ασφάλεια του συστήματος. Κάθε ζευγάρι χρηστών του συστήματος που επικοινωνεί πρέπει να μοιράζεται ένα διαφορετικό κλειδί. Έτσι ο αριθμός των κλειδιών που απαιτείται αυξάνεται σύμφωνα με το τετράγωνο του αριθμού των μελών του συστήματος. Αυτό έχει ως συνέπεια να χρειάζεται πολύ γρήγορα, πολύπλοκη διαχείριση των κλειδιών για να παραμείνουν μυστικά και ασφαλή.

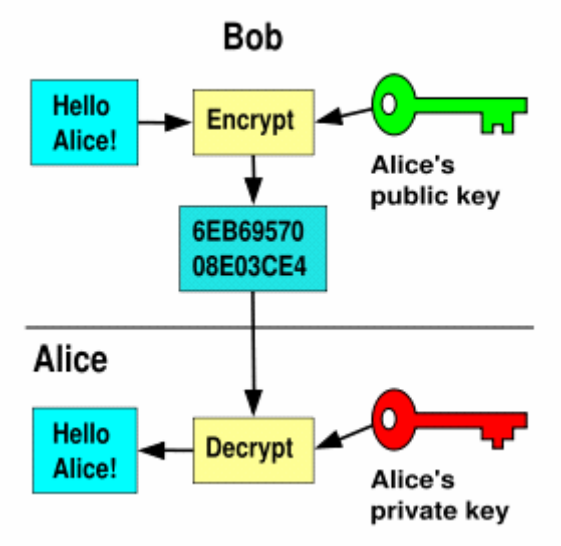
Σε μια σημαντική δημοσίευση το 1976 οι, Whitfield Diffie και Martin Hellman πρότειναν την έννοια της κρυπτογραφίας του δημόσιου κλειδιού ή αλλιώς της κρυπτογραφίας του ασύμμετρου κλειδιού κατά την οποία δύο διαφορετικά αλλά μαθηματικώς συσχετιζόμενα κλειδιά χρησιμοποιούνται. Αυτά τα δυο κλειδιά λέγονται δημόσιο *(public key*) κλειδί και ιδιωτικό κλειδί *(private key ή secret key)*. Το δημόσιο κλειδί κατασκευάζεται κατά τέτοιον τρόπο ώστε να είναι υπολογιστικά αδύνατο να προκύψει από αυτό το ιδιωτικό κλειδί, ακόμη

και αν υπάρχει κάποια αναγκαία σχέση μεταξύ τους. Άντ' αυτού και τα δυο κλειδιά παράγονται κατά μυστικό τρόπο ως ένα αλληλοσχετιζόμενο ζευγάρι.

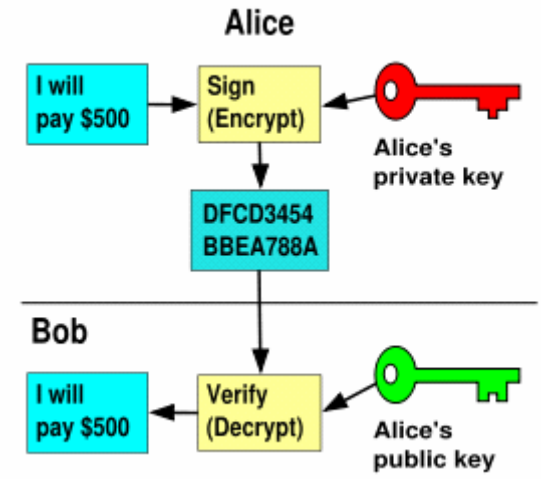
Στα συστήματα της κρυπτογραφίας δημόσιου κλειδιού, το δημόσιο κλειδί μπορεί να διανέμεται ελεύθερα μέσω διαφόρων μηχανισμών(π.χ. με *e-mail* ή με δημοσίευση σε κάποιο site), ενώ το ιδιωτικό κλειδί πρέπει να παραμένει κρυφό. Τόσο το ιδιωτικό όσο και το δημόσιο χρησιμοποιούνται και για κρυπτογράφηση αλλά και για αποκρυπτογράφηση. Συνήθως βέβαια χρησιμοποιείται το ιδιωτικό για κρυπτογράφηση και το δημόσιο για αποκρυπτογράφηση. Ακόμη, αν κάποιος κρυπτογραφήσει δεδομένα με το δημόσιο κλειδί μόνο ο κάτοχος του ιδιωτικού κλειδιού μπορεί να αποκρυπτογραφήσει τα δεδομένα και το αντίστροφο [39](βλέπε σχήματα). Οι Diffie και Hellman έδειξαν ότι αυτό το είδος της κρυπτογραφίας είναι δυνατό παρουσιάζοντας το πρωτόκολλο ανταλλαγής κλειδιών Diffie και Hellman. Το 1978 οι Ronald Rivest, Adi Shamir, και Len Adleman επινόησαν το *RSA*, ένα άλλο σύστημα κρυπτογραφίας δημοσίου κλειδιού. Το 1997 έγινε τελικά δημοσίως γνωστό ότι η ασύμμετρη κρυπτογραφία επινοήθηκε από τον James H. Ellis στην GCHQ στις αρχές τις δεκαετίας του 1970 και ότι τόσο ο αλγόριθμος Diffie-Hellman όσο και ο *RSA* είχαν αναπτυχθεί πρωτύτερα από τους Malcolm J. Williamson και Clifford Cocks αντίστοιχα. Ωστόσο οι Diffie-Hellman και οι *RSA* είναι οι πρώτοι που δημοσίευσαν υψηλής ποιότητας παραδείγματα της κρυπτογραφίας δημοσίου κλειδιού. Μερικά άλλα παραδείγματα ασύμμετρης κρυπτογραφίας είναι του *Cramer-Shoup*, η κωδικοποίηση *ElGamal*, και διάφορες τεχνικές ελλειψοειδών καμπυλών. Ακολουθούν μερικά σχήματα που παρουσιάζουν συνοπτικά τα προαναφερθέντα:



Ένας μεγάλος τυχαίος αριθμός χρησιμοποιείται για την παραγωγή του δημόσιου και του ιδιωτικού κλειδιού.



Η ασφάλεια εξαρτάται από την μυστικότητα του ιδιωτικού κλειδιού.



2.1.2.2.3 Κρυπτογραφικές συναρτήσεις κατακερματισμού (Cryptographic hash functions)

Οι κρυπτογραφικές συναρτήσεις κατακερματισμού *(Cryptographic hash functions)* ή αλλιώς συναρτήσεις κωδικοποίησης μηνυμάτων *(message digest functions)* δεν χρησιμοποιούν κλειδιά αλλά είναι ένας σχετικός και σημαντικός κλάδος των αλγορίθμων κρυπτογράφησης . Αυτές αποτελούν ένα είδος αθροίσματος ελέγχου *(checksum)*. Η έξοδος μιας κρυπτογραφικής συνάρτησης κατακερματισμού *H()* είναι πάντα του ίδιου μήκους ανεξαρτήτως του μήκους της εισόδου της. Μια δυνατή κρυπτογραφική συνάρτηση κατακερματισμού θα πρέπει για μια πολύ μικρή αλλαγή της εισόδου της να παρουσιάζει μια τελείως διαφορετική έξοδο, έτσι ώστε να είναι πραγματικά δύσκολο να συμπεράνει κανείς την είσοδο από την έξοδο. Σημαντική ιδιότητα των *hash function* αποτελεί το γεγονός ότι δεν αντιστρέφονται. Δηλαδή

δεν υπάρχει τρόπος κάποιος να βρει μια είσοδο που να αντιστοιχεί σε ένα κατακερματισμό που ήδη έχει.

Παράδειγμα: Ο αλγόριθμος *MD5* έχει έξοδο 128 bit και κρυπτογραφικά είναι πολύ δυνατός. Κατά μέσον όρο θα πρέπει κάποιος να δοκιμάσει 2^{127} εισόδους πριν να αρχίσει ο αλγόριθμος να του δίνει ίδιες εξόδους. Ακόμα όμως και αν το καταφέρει αυτό είναι μόνο μια περίπτωση από τις άπειρες εισόδους που θα μπορούσαν να παράξουν το συγκεκριμένο κατακερματισμό μηνύματος.

2^{127} = 170.141.183.460.469.231.731.687.303.715.884.105.728

Όμως με την εξέλιξη της σύγχρονης τεχνολογίας ένας χώρος αναζήτησης της τάξης του 2^{128} δεν είναι πλέον επαρκώς μεγάλος. Οι ερευνητές έχουν δημοσιεύσει αποτελέσματα συγκρούσεων, όπου κατάφεραν χρησιμοποιώντας σύγχρονη τεχνολογία και βρήκαν δύο εισόδους που είχαν τον ίδιο *MD5* κατακερματισμό καθώς και μια δεύτερη είσοδο που έχει τον ίδιο *MD5* κατακερματισμό με ένα δοσμένο *MD5* κατακερματισμό. Για αυτό δημιουργήθηκε ένας άλλος αλγόριθμος κατακερματισμού, ο *SHA-1* ο οποίος εμφανίζει έξοδο 160-bit και προς το παρόν θεωρείται περισσότερο ασφαλής.

2^{160} = 1.461.501.637.330.902.918.203.684.832.716.283.019.655.932.542.976

Vs

2^{128} = 340.282.366.920.938.463.463.374.607.431.768.211.456

Πρόσφατες όμως δημοσιεύσεις από τους Xiaoyun Wang, Yiqun Yin, και Hongbo Yu στο συνέδριο Crypto 2005 παρουσίασαν επιθέσεις εναντίον και του *SHA-1* και μάλιστα σε μεταγενέστερες αυτών οι ίδιοι ανακοίνωσαν ότι τελικά η χρονική πολυπλοκότητα είναι ακόμη μικρότερη από ότι είχαν αρχικά ανακοινώσει. Όμως το συμπέρασμα ότι ο *SHA-1* έχει τελικά "σπαστεί", είναι σχετικό αφού το να βρει κάποιος μια σύγκρουση στον 160-bit χώρο αναζήτησης του *SHA-1* ισοδυναμεί με το εξαντλητικό ψάξιμο 2^{80} συνδυασμών. Οι παραπάνω ερευνητές έδειξαν τελικά ότι αυτό γίνεται σε λιγότερα βήματα και για την ακρίβεια σε 2^{63}. Άρα το συμπέρασμα είναι ότι παρόλο που τελικά ο *SHA-1* είναι πιο αδύναμος απ' ότι είχαμε υπολογίσει αρχικά, ο αριθμός 2^{63} παραμένει αρκετά μεγάλος για να παράσχει σχετική και πρακτική ασφάλεια δεδομένου ακόμη και της σύγχρονης τεχνολογίας. Παρ' όλα αυτά οι τελευταίες έρευνες επικεντρώνονται γύρω από την ανάπτυξη αλγορίθμων με ακόμη μεγαλύτερο μέγεθος εξόδων όπως οι *SHA-256* και *SHA-512* οι οποίοι εγγυούνται ακόμη μεγαλύτερη ασφάλεια.

Cryptographic Hash Functions		
Algorithm	***Output Length***	***Notes***
MD2	128	Flawed...
MD4	128	Flawed...
MD5	128	
SHA-1	160	"Secure Hash Algorithm", but use SHA-256 or SHA-512
RMD160	160	
SHA-256	256	
Snefru	256	
GOST	256	*Gosudarstvennyi Standard Soyuza SSR*
EKS-Blowfish	448	
SHA-512	512	

Εν κατακλείδι, μπορούμε να πούμε ότι οι συγκρούσεις είναι απίθανο να συμβούν τυχαία, και πολύ δύσκολο να βρεθούν από κάποιο, αλλά δεν μπορεί κανείς να αποκλείσει αυτήν την πιθανότητα. Παραπάνω παρατίθεται ένας πίνακας που παρουσιάζει συνοπτικά μερικούς από τους αλγόριθμους κατακερματισμού που έχουν αναπτυχθεί.

2.1.2.2.4 Ψηφιακές υπογραφές (digital signatures)

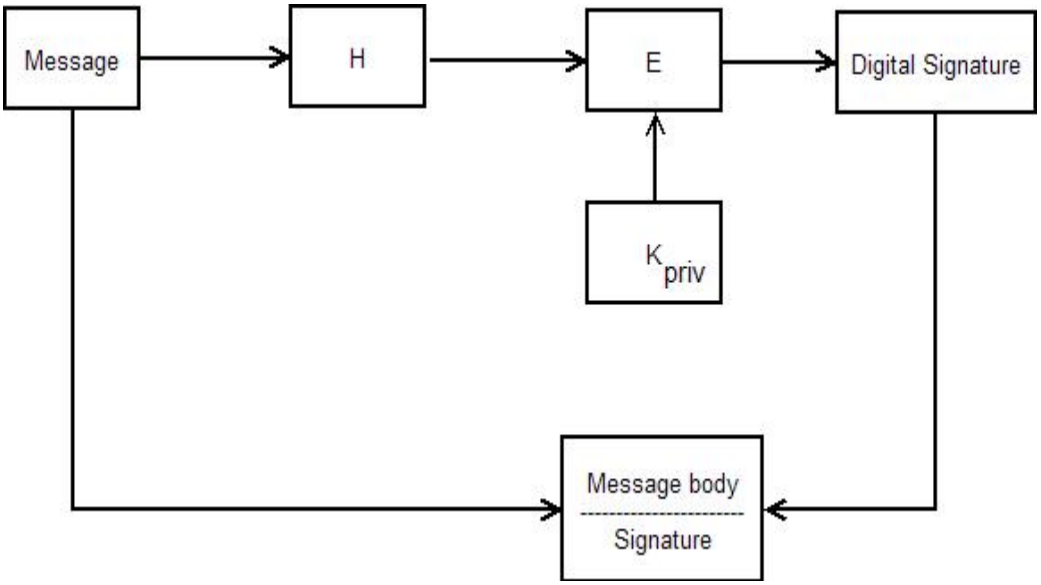
Εκτός από την κωδικοποίηση, η κρυπτογράφηση δημόσιου κλειδιού μπορεί να χρησιμοποιηθεί και για την υλοποίηση των ψηφιακών υπογραφών *(digital signature)*. Μια ψηφιακή υπογραφή μοιάζει με μια συνηθισμένη υπογραφή. Το κοινό τους χαρακτηριστικό είναι ότι είναι εύκολο να παραχθούν από ένα εξουσιοδοτημένο χρήστη αλλά δύσκολο να τις αντιγράψει κάποιος μη εξουσιοδοτημένος. Οι ψηφιακές υπογραφές είναι άρρηκτα συνδεδεμένες με το περιεχόμενο του μηνύματος που υπογράφουν. Έτσι δεν μπορούν να μεταφερθούν από ένα έγγραφο σε ένα άλλο αφού αυτό γίνεται εύκολα αντιληπτό. Η βασική ιδέα των ψηφιακών υπογράφων είναι ότι υπάρχουν δύο αλγόριθμοι, ένας για την υπογραφή των δεδομένων και ένας για την επικύρωση της υπογραφής. Κατά την υπογραφή, κωδικοποιείται ο κατακερματισμός του μηνύματος με το ιδιωτικό κλειδί του αποστολέα, ενώ κατά την επικύρωση χρησιμοποιείται το δημόσιο κλειδί με το περιεχόμενο του μηνύματος. Οι αλγόριθμοι *DSA* και *RSA* είναι δύο από τους πιο γνωστούς και δημοφιλής αλγορίθμους ψηφιακών υπογραφών. Οι ψηφιακές υπογραφές παίζουν βασικό ρόλο σε πολλά πρωτόκολλα ασφαλείας δικτύων (π.χ. *SSL / TLS*).

Οι αλγόριθμοι δημοσίου κλειδιού κυρίως βασίζονται στην πολυπλοκότητα "hard" προβλημάτων από την θεωρία αριθμών. Η hardness του RSA βασίζεται στην παραγοντοποίηση ακεραίων, ενώ του Diffie-Hellman και του *DSA* βασίζεται στο πρόβλημα του διακριτού λογαρίθμου. Πιο πρόσφατα έχει αναπτυχθεί η κρυπτογραφία της ελλειψοειδής

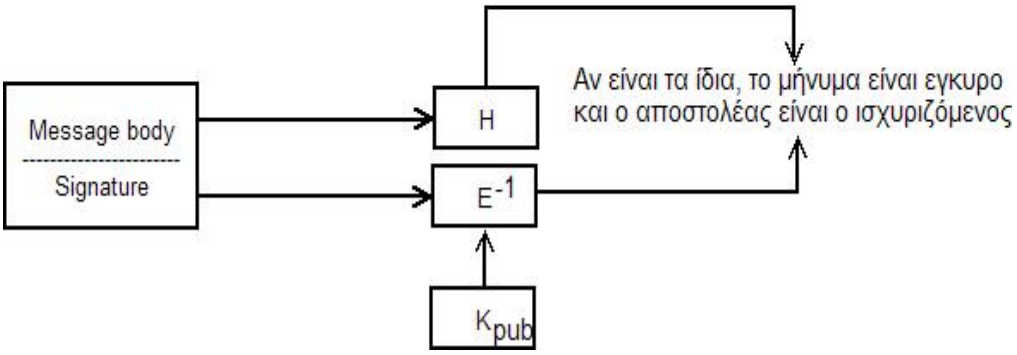
καμπύλης στην οποία η ασφάλεια βασίζεται σε προβλήματα της θεωρίας αριθμών που περιλαμβάνουν ελλειψοειδής καμπύλες.

Παρακάτω φαίνονται μερικά σχήματα που αναπαριστούν τα προαναφερθέντα.

Υπογραφή μηνύματος:



Επικύρωση μηνύματος:



όπου:

Ε ο αλγόριθμος κρυπτογράφησης που εφαρμόζεται στις αρχικές πληροφορίες και προκύπτει το κρυπτογράφημα.

Η η εφαρμογή της συνάρτησης κατακερματισμού.

E^{-1} ο αλγόριθμος αποκρυπτογράφησης που εφαρμόζεται στο κρυπτογράφημα και προκύπτουν οι αρχικές πληροφορίες.

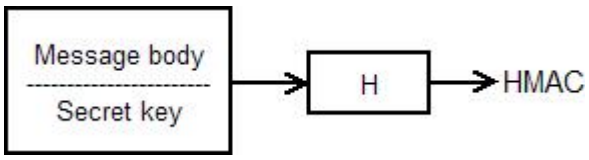
K_{priv} το ιδιωτικό κλειδί του αποστολέα

k_{pub} το δημόσιο κλειδί του αποστολέα

2.1.2.2.5 Κατακερματισμένος κωδικός αυθεντικότητας μηνυμάτων (Hashed Message Authentication Code ή HMAC)

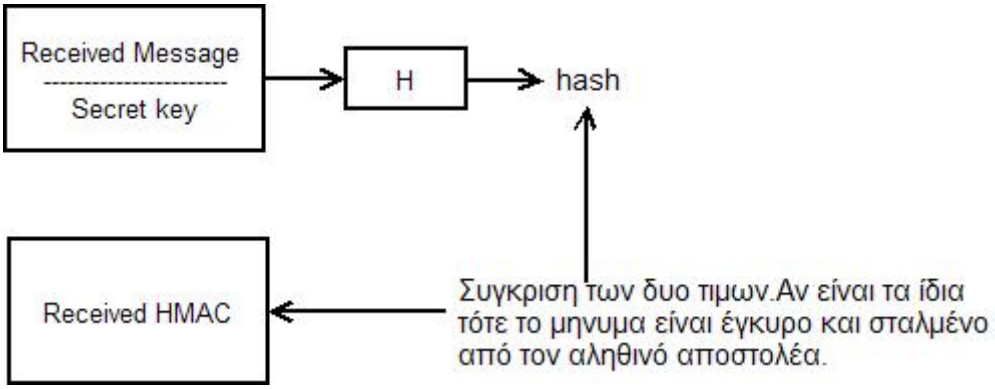
Ο αποστολέας και ο παραλήπτης μοιράζονται ένα κρυφό κλειδί *(private)*. Ο *HMAC* υπολογίζεται από τον τύπο *hash(message+secretkey)* όπου *message* είναι το μήνυμα που θέλουμε να κωδικοποιήσουμε. Ο παραλήπτης εφαρμόζει την ίδια διαδικασία *hash(message+secretkey)* προκειμένου να επικυρώσει την αυθεντικότητα του μηνύματος αλλά και ότι δεν έχει χαθεί κάποιο τμήμα του κατά την μετάδοση του *(authentication, integrity)*.

Δημιουργία HMAC:



Μετάδοση μηνύματος και του HMAC στους παραλήπτες

Επικύρωση HMAC:



Σημειώνουμε ότι μια ψηφιακή υπογραφή απαιτεί κατακερματισμό ολόκληρου του μηνύματος ακολουθούμενη από την κωδικοποίηση του κατακερματισμού. Εάν κάποιος στέλνει μόνο ένα

μεγάλο μήνυμα, η απαιτούμενη επεξεργασία υπερβαίνει αυτή της κωδικοποίησης και ουσιαστικά δεν υπάρχει κανένα πραγματικό πλεονέκτημα για τον *HMAC*. Ο *HMAC* χρειάζεται μόνο τον κατακερματισμό του μηνύματος για να παραχθεί. Εάν στέλνει κάποιος ένα μεγάλο αριθμό μηνυμάτων μικρού μεγέθους, τότε ο *HMAC* έχει υπολογιστικό πλεονέκτημα. Επομένως συμφέρει να υπογράφουμε ένα *e-mail* , αλλά να χρησιμοποιούμε τον HMAC για να επικυρώσουμε την αυθεντικότητα των *IP datagrams* πακέτων.

2.1.2.2.6 Σύγκριση συμμετρικής και ασύμμετρης κρυπτογραφίας

Σε πολλά συστήματα κρυπτογραφίας οι αλγόριθμοι κρυπτογράφησης και αποκρυπτογράφησης διαφέρουν έστω και λίγο. Σε άλλα πάλι είναι ίδιοι. Όμως αυτό το χαρακτηριστικό δεν αποτελεί την ειδοποιό διαφορά τους έτσι ώστε να μπορεί να τα χαρακτηρίσει κανείς ως συστήματα συμμετρικής ή ασύμμετρης κρυπτογραφίας.

Στα συμμετρικά συστήματα κρυπτογραφίας το κλειδί κρυπτογράφησης είναι το ίδιο με το κλειδί αποκρυπτογράφησης και μάλιστα διαφορετικό για το κάθε σύστημα ανάλογα με τις ανάγκες του. Όμως ο διαχειριστής του συστήματος πρέπει να επιλέγει με προσοχή, βάση κάποιων κανόνων τυχαιότητας της επιλογής του, το κλειδί ώστε να μην είναι εύκολα προβλέψιμο και να μεγιστοποιείται η ασφάλεια του συστήματος.

Στα ασύμμετρα συστήματα κρυπτογραφίας η κρυπτογράφηση γίνεται συνήθως με το ιδιωτικό κλειδί του αποστολέα και η αποκρυπτογράφηση με το δημόσιο. Τα δυο κλειδιά παρόλο που σχετίζονται, συνήθως μέσω μιας μαθηματικής σχέσης, δεν πρέπει να θυμίζει το ένα το άλλο αλλά να είναι τελείως διαφορετικά. Από την επιλογή αυτής της σχέσεως εξαρτάται η προβλεψιμότητα του ενός κλειδιού από το άλλο και κατά συνέπεια και η ασφάλεια του συστήματος.

2.1.2.2.7 Ψευδοτυχαίες συναρτήσεις (pseudorandom functions ή PRFs)

Μια οικογένεια συναρτήσεων $\{f_k\}_{k \in \{0,1\}^m}$ (όπου m είναι το μήκος του κλειδιού που επιλέχθηκε ως παράμετρος ασφαλείας), είναι μια οικογένεια ψευδοτυχαίων συναρτήσεων εάν ένας εισβολέας A (του οποίου οι πόροι είναι περιορισμένοι σε πολυωνυμικό χρόνο m) δεν μπορεί να διακρίνει τη διαφορά μεταξύ μιας συνάρτησης f_k (της οποίας το k είναι επιλεγμένο τυχαία και κρυφό) και μιας πραγματικής τυχαίας συνάρτησης με αμελητέα πιθανότητα πρόβλεψης. Δηλαδή μια συνάρτηση g θα ανήκει σ' αυτήν την οικογένεια αν θα μπορούσε ισοδύναμα να επιλεχθεί ως g μια συνάρτηση f_k με τυχαίο κλειδί k μήκους m bits ή μια πραγματική συνάρτηση με το ίδιο εύρος τιμών.

Για τις ψευδοτυχαίες συναρτήσεις ισχύει η ιδιότητα: Όσο το κλειδί *k* παραμένει τυχαίο (ή ψευδοτυχαίο) και κρυφό, η τιμή $k_1=f_k(x)$ είναι επίσης ψευδοτυχαία για οποιοδήποτε γνωστό x. Αυτό μας επιτρέπει να επαναλαμβάνουμε τον αλγόριθμο με ασφάλεια. Δηλαδή η $k_2=f_{k1}(x)$ είναι επίσης ψευδοτυχαία και ούτω κάθε εξής. Επιπλέον η τιμή $k_1'=f_k(x')$ όπου $x \neq x'$ είναι κρυπτογραφικώς ανεξάρτητη από τη k_1 (όσο παραμένει η *k* κρυφή) και μπορεί να χρησιμοποιηθεί σαν κλειδί για διάφορους κρυπτογραφικούς μετασχηματισμούς (π.χ. *MAC*)

2.2 Συστήματα εκπομπής δεδομένων

2.2.1 Εισαγωγή

Υπάρχουν τρία μοντέλα επικοινωνίας υπολογιστών σε ένα δίκτυο, το μοντέλο unicast, το μοντέλο broadcast και το μοντέλο multicast. Τα τρία αυτά μοντέλα θα παρουσιαστούν στη συνέχεια

2.2.2 Μοντέλο unicast

Μοντέλο unicast

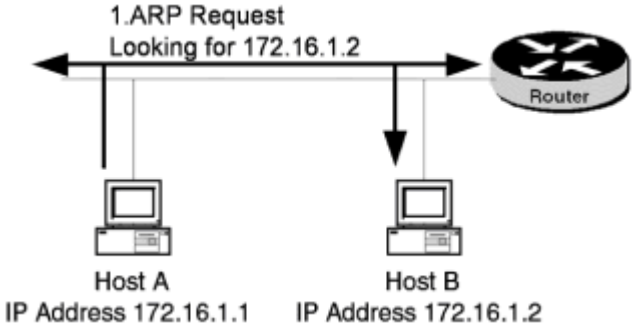
Το πρώτο μοντέλο είναι το *unicast* το οποίο αφορά επικοινωνία ένας προς ένα. Για να πραγματοποιηθεί η αποστολή των δεδομένων σε ένα τοπικό δίκτυο *(LAN)* ο αποστολέας χρειάζεται να γνωρίζει τόσο την *IP* διεύθυνση του παραλήπτη, όσο και την *ethernet* διεύθυνση του. Η *ethernet* διεύθυνση δημιουργείται στην κάρτα διεπαφών δικτύου *(Network Interface Card, NIC)* και είναι ένας αριθμός 48 bit που χαρακτηρίζει τον υπολογιστή με μοναδικό τρόπο, ο οποίος χρησιμοποιείται από το στρώμα διασύνδεσης *(link layer)* του μοντέλου *OSI*. Ο αριθμός αυτός δεν αλλάζει παρά μόνο αν αλλάξει η κάρτα διεπαφών δικτύου. Ξέρουμε ότι σε ένα τοπικό δίκτυο όλες οι πληροφορίες ανταλλάσσονται με τη μορφή ethernet πλαισίων, επομένως τα σταλθέντα πακέτα *IP* θα πρέπει να έρθουν στη μορφή αυτών. Για να συμπληρωθεί η μορφοποίηση τους ο αποστολέας θα πρέπει να αναλύσει την *IP* διεύθυνση του παραλήπτη σύμφωνα με τη χαρτογράφηση διευθύνσεων *ethernet*. Η χαρτογράφηση ολοκληρώνεται χρησιμοποιώντας το πρωτόκολλο ανάλυσης διευθύνσεων *ARP*.

Στα σχήματα ένας υπολογιστής *A* επιθυμεί να στείλει ένα πακέτο στον υπολογιστή *B*. Ο *A* γνωρίζει την *IP* διεύθυνση του *B* αλλά όχι και την *ethernet* διεύθυνση του. Η διαδικασία που ακολουθεί το πρωτόκολλο *ARP* έχει ως εξής:

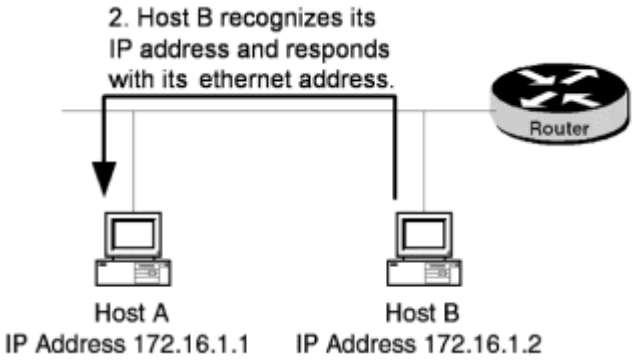
1. Ο *A* μεταδίδει σε όλους τους υπολογιστές του υποδικτύου συμπεριλαμβανομένου του router μια *ARP request*. (Σχήμα 1)

2. Ο *B* λαμβάνει την *ARP request* και αναγνωρίζει ότι η *IP* διεύθυνση που περιέχει είναι η δική του. Κατόπιν απαντάει στον *A* στέλνοντας μια *ARP reply* που περιέχει την *ethernet* διεύθυνση του. (Σχήμα 2)

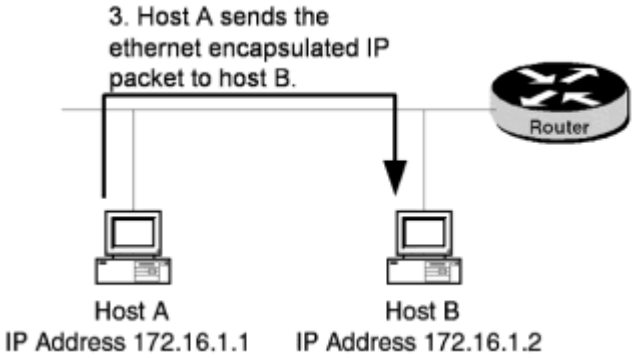
3. Ο *A* λαμβάνει την απάντηση του *B* και μπορεί να αρχίσει την αποστολή δεδομένων στον *B* κατασκευάζοντας πλαίσια *ethernet*. (Σχήμα 3)



***Σχήμα 2-1*:**Ανάλυση της IP με χαρτογράφηση διευθύνσεων ethernet βήμα 1



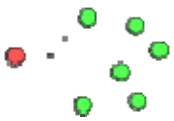
Σχήμα 2-2:Ανάλυση της IP με χαρτογράφηση διευθύνσεων ethernet βήμα 2



Σχήμα 2-3:Ανάλυση της IP με χαρτογράφηση διευθύνσεων ethernet βήμα 3

Αν ένας υπολογιστής *A* επιθυμεί να στείλει ένα πακέτο σε έναν υπολογιστή *B* ο οποίος δεν ανήκει στο ίδιο υποδίκτυο *IP* , τότε το πακέτο πρέπει να σταλεί στον router του δικτύου μέσω της *gateway* του. Η *gateway* δείχνει τη διεπαφή του *router* που συνδέεται με το τοπικό δίκτυο, όπου ανήκει ο αποστολέας *A*. Επειδή ο παραλήπτης *B* είναι σε διαφορετικό υποδίκτυο, ο αποστολέας *A* στέλνει ένα πλαίσιο στον *router* το οποίο περιέχει το πακέτο *IP*. Όταν ο *router* λάβει το πλαίσιο, το πακέτο *IP* εξάγεται και ο *router* καθορίζει από τη διεύθυνση προορισμού *IP* εάν ο προορισμός είναι ή όχι σε ένα άμεσα συνδεδεμένο δίκτυο. Εάν είναι σε ένα άμεσα συνδεδεμένο δίκτυο, ο *router* στέλνει μια *ARP request* σε εκείνο το δίκτυο για να αναλύσει τη διεύθυνση *ethernet* του προορισμού. Όταν ο *router* λάβει την απάντηση *ARP*, κατασκευάζει ένα πλαίσιο *ethernet* που περιέχει το πακέτο IP και έπειτα στέλνει το πλαίσιο στον προορισμό. Εάν ο προορισμός δεν είναι σε ένα άμεσα συνδεδεμένο δίκτυο, ο router συμβουλεύεται τον πίνακα δρομολόγησης και καθορίζει τον επόμενο router όπου το πλαίσιο πρέπει να σταλεί. Τα πρωτόκολλα δρομολόγησης *IP unicast* δεν καλύπτονται στην παρούσα εργασία, αλλά οι αναφορές παρατίθενται στο τέλος αυτής για περαιτέρω μελέτη.

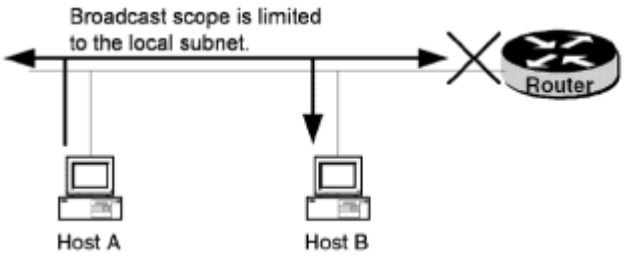
2.2.3 Μοντέλο broadcast



Μοντέλο broadcast

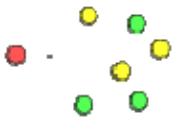
Το μοντέλο *broadcast* αναφέρεται στην επικοινωνία ένας προς όλους. Όταν ένας υπολογιστής στέλνει δεδομένα σε μια *broadcast Ethernet* διεύθυνση (συνήθως είναι η 0x FF FF FF FF FF FF) τότε όλοι οι υπολογιστές που ανήκουν σε αυτό το υποδίκτυο λαμβάνουν αυτά τα δεδομένα μηδενός εξαιρουμένου. Η broadcast μετάδοση δεδομένων περιορίζεται στο

επίπεδο του υποδικτύου. Πρακτικά αυτό σημαίνει ότι η μετάδοση μπλοκάρει στο *router* του δικτύου.



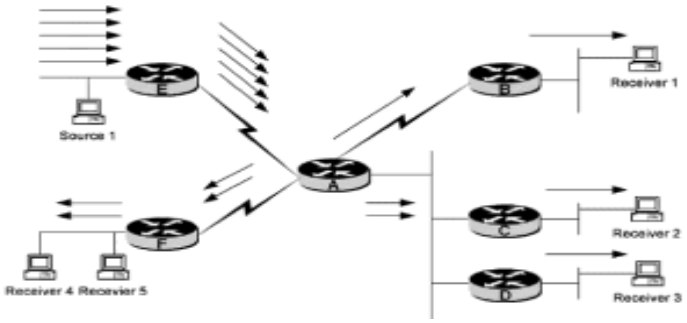
Σχήμα 2-4

2.3 Μοντέλο multicast



Μοντέλο multicast

Το ερώτημα που προκύπτει από την μελέτη των δυο προηγούμενων πρωτοκόλλων είναι εάν ένας υπολογιστής θέλει να στείλει το ίδιο πακέτο σε περισσότερους από έναν δέκτες, πώς μπορεί αυτό να ολοκληρωθεί; Θα μπορούσαμε να προσπαθήσουμε το πρότυπο επικοινωνίας unicast και θα ήμασταν επιτυχείς, αν δεν υπήρχαν μερικά ουσιαστικά προβλήματα. Έστω ότι ο υπολογιστής *A* θέλει να στείλει ένα πακέτο σε πέντε υπολογιστές χρησιμοποιώντας το πρότυπο *unicast*. Αυτό υπονοεί ότι ο *A* ξέρει τη διεύθυνση *IP* κάθε δέκτη. Εάν αυτό συμβαίνει, κατόπιν ο *A* θα πρέπει να στείλει το ίδιο πακέτο σε πέντε διαφορετικές διευθύνσεις *IP*, όπως φαίνεται στο σχήμα 2-6.



Σχήμα 2-5: Χρήση του μοντέλου επικοινωνίας unicast για να επιτύχει multicasting ιδιότητες

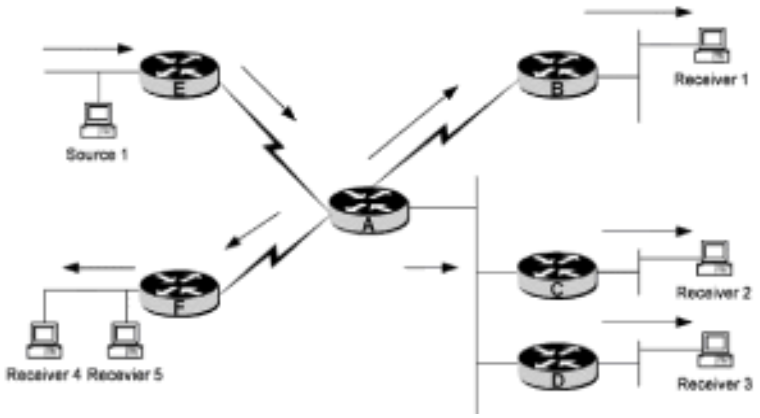
Δεδομένου ότι ο αριθμός δεκτών αυξάνεται, ο αριθμός των πακέτων που θα πρέπει να σταλεί θα αυξάνεται γραμμικά. Με άλλα λόγια, για *ν* δέκτες, ο *Α* θα πρέπει να στείλει *ν* αντίγραφα κάθε πακέτου. Αυτό όμως αποτελεί σοβαρό πρόβλημα. Για παράδειγμα αν ο *Α* στέλνει μια ακουστική ή τηλεοπτική παρουσίαση σε πραγματικό χρόνο, η λύση αυτή μπορεί να είναι εφαρμόσιμη για πολύ λίγους δέκτες. Στο μεταξύ όσο θα αυξάνει ο αριθμός των δεκτών, το φορτίο της αντιγραφής των πακέτων στον υπολογιστή *Α* θα είναι τέτοιο που η καθυστέρηση μεταξύ των πακέτων θα γίνει πάρα πολύ μεγάλη. Επίσης, οι συνδέσεις στο *router* της πηγής, στο σχήμα 2-5 είναι ο *router E*, θα μείωναν σοβαρά το εύρος ζώνης *(bandwidth)*.

Ένα άλλο σημαντικό πρόβλημα με αυτήν την προσέγγιση είναι ότι ο υπολογιστής *Α* δεν ξέρει που είναι οι παραλήπτες. Ακόμη εάν ο αριθμός των παραληπτών δεν μεταβάλλεται, τότε δεν θα μπορούν να εισαχθούν κι άλλοι παραλήπτες. Όμως αυτό είναι εξαιρετικά περιοριστικό αφού νέοι υπολογιστές δεν θα μπορούν να γίνουν μέλος της ομάδας των παραληπτών. Επί πλέον οι ήδη υπάρχοντες στην ομάδα παραλήπτες, δεν θα μπορούν να αποχωρήσουν από αυτή.

Βεβαίως θα μπορούσε κανείς να προτείνει το μοντέλο *broadcast*. Σ' αυτήν την περίπτωση κάθε υπολογιστής στο τοπικό υποδίκτυο θα ελάμβανε την κυκλοφορία και κάθε πακέτο θα έπρεπε μόνο να σταλεί μία φορά. Τα προβλήματα που ελλοχεύουν είναι δύο. Το πρώτο είναι ότι μόνο οι παραλήπτες στο ίδιο υποδίκτυο λαμβάνουν την κυκλοφορία, ενώ οι παραλήπτες σε άλλα υποδίκτυα δεν μπορούν να την λάβουν επειδή ο *router* εμποδίζει τη μετάδοση. Το δεύτερο πρόβλημα είναι ότι κάθε παραλήπτης πρέπει να επεξεργάζεται την καθολική μετάδοση των πακέτων που υπάρχουν στο δίκτυο προκειμένου να καθορίσει εάν προορίζονται για αυτόν. Το πακέτο *IP* θα έπρεπε να εξάγεται κάθε φορά από το πλαίσιο *ethernet* και, επειδή η διεύθυνση προορισμού *IP* είναι επίσης μια διεύθυνση broadcast μετάδοσης, το μέρος *UDP* ή *TCP* του πακέτου θα πρέπει να εξαχθεί και να περάσει επάνω στη λίστα πρωτοκόλλου. Εάν υπάρχει μια διαδικασία που αναμένει τα δεδομένα, τότε τα δεδομένα περνάνε στο στρώμα εφαρμογής *(application layer)*. Εάν δεν υπάρχει τότε τα δεδομένα απορρίπτονται. Για εκείνους τους υπολογιστές (ή εφαρμογές) που δεν αναμένουν δεδομένα, αυτό θα ήταν χάσιμο πολύτιμου χρόνου επεξεργασίας και αιτία παραπόνων για πολλούς χρήστες. Επομένως φαίνεται να γεννιέται η ανάγκη δημιουργίας ενός νέου μοντέλου επικοινωνίας, το μοντέλο *multicast*.

Για το μοντέλο *multicast*, θα χρειαστούμε δύο νέους τύπους διευθύνσεων, τον τύπο διευθύνσεων *multicast IP* και τον τύπο διευθύνσεων *multicast ethernet*. Μια διεύθυνση *multicast IP* προσδιορίζει μια ομάδα παραληπτών που θέλουν να λάβουν τα δεδομένα που στέλνονται. Επειδή όλα τα πακέτα *IP* είναι διαμορφωμένα σε πλαίσια *ethernet*, χρειάζεται και μια διεύθυνση *multicast ethernet*. Για να λειτουργήσει το μοντέλο *multicast* σωστά, οι υπολογιστές πρέπει να είναι σε θέση να λάβουν και το *unicast* και το *multicast* φορτίο δεδομένων, πράγμα που σημαίνει ότι οι υπολογιστές χρειάζονται πολλαπλάσιες IP και

ethernet διευθύνσεις. Οι *unicast IP* και *ethernet* διευθύνσεις χρησιμοποιούνται για το *unicast* φορτίο δεδομένων και μηδενικές ή περισσότερες *multicast* διευθύνσεις IP και *ethernet*, χρησιμοποιούνται για το *multicast* φορτίο δεδομένων. Δεν χρειάζεται καμία *multicast* διεύθυνση εάν ο αποστολέας δεν θα είναι συγχρόνως και παραλήπτης. Ένα ζευγάρι των multicast διευθύνσεων, *IP* και *ethernet*, απαιτείται για κάθε *multicast* ομάδα για την οποία κάποιος παραλήπτης επιθυμεί να γίνει μέλος. Μια σημαντική διαφορά μεταξύ του *unicast* και των *multicast* διευθύνσεων είναι ότι *unicast* διευθύνσεις είναι μοναδικές σε κάθε υπολογιστή, ενώ οι *multicast* διευθύνσεις δεν είναι. Εάν πέντε υπολογιστές επιθυμούν να λάβουν το *multicast* φορτίο δεδομένων που προορίζεται για την ομάδα Α παραδείγματος χάριν, τότε οι οικοδεσπότες όλοι θα αφουγκράζονταν την κυκλοφορία που προορίζεται για την ίδια το *multicast* διεύθυνση, *IP* και *ethernet*. Όπως φαίνεται στο σχήμα 2-6 το φορτίο δεδομένων από την περίπτωση *unicast* μειώνεται πολύ.



Σχήμα 2-6:Το μοντέλο επικοινωνίας *multicast*

3

Το πρωτόκολλο αυθεντικοποίησης TESLA

3.1 Εισαγωγή

Στις μέρες μας παρατηρούμε την ολοένα και μεγαλύτερη αύξηση του αριθμού των χρηστών του διαδικτύου. Αποτέλεσμα της αύξησης αυτής, είναι να μεγαλώνει συνεχώς και το φορτίο των δεδομένων (για παράδειγμα ρεύματα δεδομένων ήχου και βίντεο) που ανταλλάσσονται μεταξύ των χρηστών με συνέπεια την μείωση της απόδοσης των υπηρεσιών του διαδικτύου. Αναμένεται η αύξηση αυτή να συνεχιστεί .

Για να επιτευχθεί όμως μια ασφαλή εκπομπή δεδομένων θα πρέπει πρώτα να μπορεί κάποιος να λάβει τις απαραίτητες εγγυήσεις ασφάλειας. Ένας ακόμη σημαντικός κίνδυνος ασφάλειας από την πλευρά του χρήστη του διαδικτύου είναι η αυθεντικότητα των δεδομένων *(data authenticity*). Ο χρήστης χρειάζεται να είναι σίγουρος ότι το ρεύμα δεδομένων προέρχεται από τον ισχυριζόμενο αποστολέα. Διαφορετικά, κάποιος εισβολέας θα μπορούσε να αντικαταστήσει μέρη του ρεύματος δεδομένων με το υλικό του. Παραδείγματος χάριν, ένας εισβολέας θα μπορούσε να αλλάξει τα δεδομένα που διανέμονται μέσω της πολυεκπομπής δεδομένων IP και να ισχυριστεί ότι αυτός είναι ο γνήσιος αποστολέας τους. Σε αυτό το σενάριο, ο δέκτης χρειάζεται την ισχυρή επαλήθευση αποστολέων και δεδομένων.

Το πρόβλημα της συνεχούς επαλήθευσης ρευμάτων δεδομένων για την περίπτωση ενός αποστολέα και ενός παραλήπτη έχει λυθεί μέσω των τυποποιημένων μηχανισμών, π.χ. [12,18]. Ο αποστολέας και ο δέκτης συμφωνούν σχετικά με ένα μυστικό κλειδί που χρησιμοποιείται από κοινού με έναν κώδικα αυθεντικοποίησης μηνυμάτων *(MAC)* για να εξασφαλίσει την αυθεντικότητα κάθε πακέτου. Όμως στην περίπτωση πολλών δεκτών το πρόβλημα γίνεται πολύ μεγαλύτερο. Η χρήση της συμμετρικής κρυπτογραφίας θα επέτρεπε στο καθένα που διαθέτει ένα κλειδί (δηλαδή οποιοδήποτε παραλήπτη) να υποκλέψει τα πακέτα. Εναλλακτικά, ο αποστολέας μπορεί να χρησιμοποιήσει τις ψηφιακές υπογραφές για

να υπογράψει κάθε πακέτο με το ιδιωτικό κλειδί του. Αυτή η λύση παρέχει επαρκή επαλήθευση, όμως οι ψηφιακές υπογραφές είναι απαγορευτικά ανεπαρκείς.

Τα ρεύματα δεδομένων πραγματικού χρόνου συνήθως εμφανίζουν απώλειες, το οποίο καθιστά το πρόβλημα ασφάλειας ακόμα μεγαλύτερο. Όταν υπάρχουν πολλοί παραλήπτες στο σύστημα, το εύρος ζώνης των παραληπτών εμφανίζει μεγάλη διακύμανση από παραλήπτη σε παραλήπτη. Πιο συγκεκριμένα οι παραλήπτες με σχετικά μικρό εύρος ζώνης εμφανίζουν την πιο μεγάλη απώλεια πακέτων. Εντούτοις, θέλουμε να επιβεβαιώσουμε την αυθεντικότητα των δεδομένων ακόμη και με την παρουσία αυτής της υψηλής απώλειας πακέτων. Διάφορα πρωτόκολλα έχουν προταθεί για την επίλυση αυτού του προβλήματος (δηλ. την επαλήθευση της αυθεντικότητας των δεδομένων και του αποστολέα σε μια κατάσταση όπου μόνο ο αποστολέας είναι έμπιστος) έχουν προταθεί στο παρελθόν [7, 13, 28, 31], αλλά κανένα από αυτά τα σχέδια δεν είναι απολύτως ικανοποιητικό.

Σε αυτήν την εργασία παρουσιάζεται το πρωτόκολλο *TESLA (Timed Efficient Stream Loss - tolerant Authentication)*. Πριν να περιγράψουμε την βασική ιδέα του *TESLA* θα πρέπει να πούμε ότι τα κλειδιά που χρησιμοποιούνται από τον αποστολέα σχηματίζουν μια μονόδρομη αλυσίδα *(one way chain)*.Οι μονόδρομες αλυσίδες και οι λειτουργίες τους περιγράφονται σε ξεχωριστή ενότητα. Η βασική ιδέα του *TESLA* έχει ως εξής: Το *TESLA* χρησιμοποιεί μόνο συμμετρικές κρυπτογραφικές θεμελιώδεις λειτουργίες όπως οι ψευδοτυχαίες συναρτήσεις *(PRFs)* και τους κώδικες αυθεντικότητας μηνυμάτων *(MACs ή message authentication codes)* ενώ παράλληλα βασίζεται στη συγχρονισμένη αποκάλυψη κλειδιών από τον αποστολέα. Πιο συγκεκριμένα ο αποστολέας δεσμεύεται σε ένα τυχαίο κλειδί k το οποίο το κρατάει κρυφό και μεταδίδει τη δέσμευση στους παραλήπτες (όχι το ίδιο το κλειδί). Ο αποστολέας στη συνέχεια προσαρτεί έναν κωδικό αυθεντικότητας μηνύματος στο επόμενο πακέτο P_i και χρησιμοποιεί το κλειδί k ως το κλειδί για να κατασκευάσει τον *MAC*. Σε ένα μεταγενέστερο πακέτο P_{i+1} ο αποστολέας αποκαλύπτει το k επομένως οι παραλήπτες μπορούν να κατασκευάσουν τν *MAC* του P_i . Εάν οι δύο *MACs* είναι ίσες και είναι εγγυημένο ότι το P_{i+1} δεν εστάλη πριν το P_i τότε ο παραλήπτης γνωρίζει σίγουρα ότι το πακέτο είναι αυθεντικό. Για την αρχικοποίηση του πρωτοκόλλου ο αποστολέας χρησιμοποιεί ένα συνηθισμένο αλγόριθμο υπογραφής για να υπογράψει το αρχικό πακέτο που στέλνει σε κάθε παραλήπτη. Το αρχικό πακέτο περιέχει το κλειδί στο οποίο δεσμεύτηκε ο αποστολέας. Όλα τα πακέτα αυθεντικοποιούνται μέσω των αλυσίδων που σχηματίζουν τα κλειδιά.

Το *TESLA* παρουσιάζει τις εξής ιδιότητες:

- *Μικρές απαιτήσεις υπολογιστικής ισχύς.* Η αυθεντικοποίηση συμπεριλαμβάνει τυπικά τόσο για τη *MAC* όσο και για τη συνάρτηση κατακερματισμού ένα υπολογισμό ανά πακέτο, τόσο για τον αποστολέα όσο και για τον παραλήπτη.
- *Χαμηλές απαιτήσεις επικοινωνίας ανά πακέτο.* Οι απαιτήσεις μπορεί να είναι χαμηλές (περίπου 10 bytes / πακέτο).

- *Η ροή των δεδομένων είναι μιας κατεύθυνσης*. Τα δεδομένα ρέουν μόνο από τον αποστολέα προς τον παραλήπτη. Μετά την εκκίνηση της επικοινωνίας δεν χρειάζεται η ανταλλαγή καμίας άλλης αναφοράς μηνυμάτων. Αυτό συνεπάγεται ότι οι απαιτήσεις αυθεντικοποίησης των δεδομένων του αποστολέα είναι ανεξάρτητο από τον αριθμό των παραληπτών, επομένως το πρωτόκολλο *TESLA* είναι πολύ ευέλικτο.
- *Ο αποστολέας δεν χρειάζεται να αποθηκεύσει προσωρινά τα πακέτα* Το κάθε πακέτο στέλνεται χωρίς καθυστέρηση μόλις είναι έτοιμο προς αποστολή.
- *Υψηλή εγγύηση αυθεντικότητας*. Το *TESLA* παρέχει αυθεντικοποίηση υψηλής ποιότητας. Με τον όρο υψηλή ποιότητας αυθεντικοποίηση εννοούμε ότι ο παραλήπτης έχει υψηλή επιβεβαίωση της αυθεντικότητας, όσο ισχύουν οι υποθέσεις του συγχρονισμού και οι κρυπτογραφικές θεμελιώδεις λειτουργίες που περιγράψαμε παραπάνω.
- *Δεδομένα συγχρονισμένης παραγωγής (freshness of data)*.Κάθε παραλήπτης γνωρίζει ένα πάνω όριο του χρόνου διάδοσης των πακέτων.

3.2 TESLA: Timed Efficient Stream Loss-tolerant Authentication

Σε αυτήν την ενότητα περιγράφουμε πέντε αλγόριθμους για αυθεντικοποίηση ρεύματος δεδομένων. Το κάθε ένα από αυτό βασίζεται πάνω στους προηγούμενους και βελτιώνει τις ελλείψεις τους. Τελικά ο πέμπτος αλγόριθμος, ο οποίος καλείται *TESLA*, ικανοποιεί όλες τις ιδιότητες που παρουσιάσαμε στην εισαγωγή. Χρησιμοποιούμε τους συμβολισμούς $<x , y>$ συμβολίζει την συνένωση των x και y, S για τον αποστολέα *(sender)* και R για τον παραλήπτη *(receiver)*. Ένα ρεύμα δεδομένων P αποτελείται από τμήματα M_i (τα οποία καλούμε μηνύματα), $P=<M_1,M_2,...,M_n>$. Κάθε μήνυμα M_i στέλνεται μέσα σε ένα πακέτο P_i, το οποίο περιέχει ακόμη και κάποιες επιπρόσθετες πληροφορίες αυθεντικοποίησης.

3.2.1 Το μοντέλο εισβολέα και εγγύηση ασφάλειας

Οι αλγόριθμοι σχεδιάστηκαν έτσι ώστε να προσφέρουν ασφάλεια απέναντι σε έναν ισχυρό εισβολέα με τις ακόλουθες ιδιότητες:

- Πλήρης έλεγχος του δικτύου. Ο εισβολέας μπορεί να υποκλέψει, να αιχμαλωτίσει, να ξαναστείλει, να σταματήσει, να καθυστερήσει και να τροποποιήσει πακέτα.
- Έχει πρόσβαση σε ένα γρήγορο δίκτυο με αμελητέα καθυστέρηση.

- Οι υπολογιστικοί πόροι του είναι μεγάλοι αλλά όχι και απεριόριστοι. Πιο συγκεκριμένα αυτό σημαίνει ότι μπορεί να πραγματοποιήσει επαρκείς υπολογισμούς, όπως ο υπολογισμός ενός λογικού αριθμού από ψευδοτυχαίες εφαρμογές και *MACs* με αμελητέα καθυστέρηση. Εντούτοις δεν μπορεί να αντιστρέψει μια ψευδοτυχαία συνάρτηση (ή να τη ξεχωρίσει από μια πραγματική τυχαία συνάρτηση) με μη αμελητέα πιθανότητα. Η ασφάλεια που εγγυάται το *TESLA* είναι ότι ο παραλήπτης δεν δέχεται σαν αυθεντικό κανένα μήνυμα M_i εκτός των M_i που έχουν σταλεί πραγματικά από τον αποστολέα. Ένας αλγόριθμος που παρέχει αυτήν την εγγύηση λέγεται ασφαλής αλγόριθμος αυθεντικοποίησης δεδομένων.

Παρ' όλα αυτά οι παραπάνω απαιτήσεις της ασφάλειας, δεν παρέχουν προστασία στην αποστολή πιστών αντιγράφων μηνυμάτων. Τέτοιου είδους προστασίες μπορεί να προστεθούν ανεξάρτητα από κάποιος άλλους μηχανισμούς, όπως οι *nonces* και οι σειριακοί αριθμοί. Οι αλγόριθμοι 1 – 3 που περιγράφονται παρακάτω παρέχουν τέτοιου είδους προστασία.

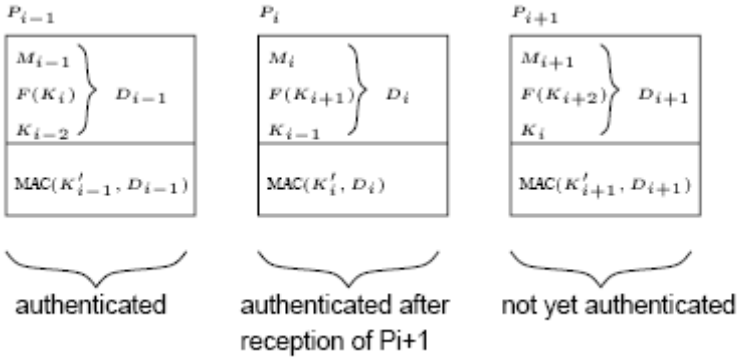
3.2.2 Αρχικός συγχρονισμός

Και οι πέντε αλγόριθμοι ξεκινούν με ένα αρχικό πρωτόκολλο συγχρονισμού όπου ο κάθε παραλήπτης συγκρίνει την τοπική ώρα του με αυτήν του αποστολέα και αποθηκεύει την διαφορά. Σημειώνουμε ότι ένα χαλαρό άνω όριο στην διαφορά των δυο ρολογιών είναι αρκετό. Στην πραγματικότητα το μόνο που χρειάζεται να ξέρει ο παραλήπτης είναι μια τιμή δ τέτοια που το ρολόι του αποστολέα να μην είναι μπροστά από το ρολόι του παραλήπτη πάνω από δ χρονικές μονάδες. Το δ μπορεί να είναι μέγεθος της τάξης αρκετών δευτερολέπτων. Στην ενότητα 3.8 περιγράφουμε ένα απλό πρωτόκολλο και αναλύουμε κάποια θέματα ευελιξίας που σχετίζονται με τον αρχικό συγχρονισμό. Μια βασική θεμελιώδης υπόθεση για την υπόθεση της ασφάλειας είναι ότι τα τοπικά εσωτερικά ρολόγια του αποστολέα και του παραλήπτη μετράνε σωστά και δεν χάνουν καθόλου.

3.2.3 Αλγόριθμος 1:Ο βασικός αλγόριθμος

Ακολουθεί μια περίληψη του βασικού αλγορίθμου *TESLA* .Ο αποστολέας εκδίδει μια υπογεγραμμένη δέσμευση σε ένα κλειδί την οποία γνωρίζει μόνο ο ίδιος. Κατόπιν ο αποστολέας χρησιμοποιεί αυτό το κλειδί για να υπολογίσει τον *MAC* για ένα πακέτο P_i και το αποκαλύπτει αργότερα στο πακέτο P_{i+1}, πράγμα το οποίο δίνει τη δυνατότητα στους παραλήπτες να επαληθεύσουν την δέσμευση και την *MAC* του πακέτου P_i. Εάν και οι δύο *MACs* είναι ίσες, το πακέτο P_i είναι αυθεντικό και ασφαλή. Η δέσμευση του αποστολέα απέναντι στα κλειδιά πραγματοποιείται μέσω μιας ψευδοτυχαίας συνάρτησης με αντίσταση συγκρούσεων *(collision resistance)*.

Κατόπιν περιγράφουμε με περισσότερες λεπτομέρειες τον βασικό αλγόριθμο. Οι βασικές ιδέες του αλγορίθμου αναπαριστούνται στο σχήμα 3.1.



Σχήμα 3-1

Βασικός αλγόριθμος αυθεντικοποίησης ρεύματος δεδομένων.

Συμβολισμοί:

M_i:μήνυμα i, P_i:πακέτο i, K_i: συμβολίζει το κρυφό κλειδί, F και F' είναι ψευδοτυχαίες συναρτήσεις, $MAC(K_i,D_i)$ συμβολίζεται ο υπολογισμός του MAC του πακέτου i χρησιμοποιώντας το κρυφό κλειδί $K_i' = F'(Ki)$.

Υποθέτουμε ότι ο παραλήπτης έχει ήδη ένα αυθεντικοποιημένο πακέτο $P_{i-1} = <D_{i-1},MAC(K_{i-1}', D_{i-1})>$ (όπου $D_{i-1}=<M_{i-1},F(K_i),K_{i-2}>$). Τα πεδία έχουν την ακόλουθη σημασία, M_{i-1} είναι το μήνυμα που περιέχει το πακέτο, $K_i'=F'(K_i)$ είναι το κρυφό κλειδί που χρησιμοποιήθηκε για τον υπολογισμό του MAC του επόμενου πακέτου και $F(K_i)$ είναι η δέσμευση του αποστολέα στο κλειδί K_i χωρίς να το αποκαλύπτει όμως. Οι συναρτήσεις F_i και F_i' είναι δύο διαφορετικές ψευδοτυχαίες συναρτήσεις. Η τιμή της δέσμευσης $F(K_i)$ είναι σημαντική για την αυθεντικοποίηση των επόμενων πακέτων P_i. Για την εκκίνηση του βασικού αλγορίθμου, το πρώτο πακέτο χρειάζεται να αυθεντικοποιηθεί με ένα συνηθισμένο αλγόριθμο ψηφιακής υπογραφής, για παράδειγμα τον *RSA*[27].

Για να στείλει το μήνυμα M_i, ο αποστολέας επιλέγει ένα καινούριο τυχαίο κλειδί K_{i+1} (εφαρμόζοντας την ψευδοτυχαία συνάρτηση ως εξής $F(K_i)=K_{i+1}$) και κατασκευάζει το επόμενο πακέτο $P_i=<D_i,MAC(K_i',D_i)>$, όπου $D_i=<M_i,F(K_{i+1}),K_{i-1}>$ και με $MAC(K_i',D_i)$ συμβολίζουμε τον υπολογισμό του κώδικα αυθεντικοποίησης του D_i από το κλειδί K_i.

Όταν ο αποστολέας λάβει το πακέτο P_i, δεν μπορεί άμεσα να ελέγξει τον MAC του αφού δεν γνωρίζει το K_i και έτσι δεν μπορεί να ξανακατασκευάσει το K_i'. Το πακέτο $P_{i+1} = <D_{i+1},MAC(K_{i+1}', D_{i+1})>$ όπου $D_{i+1} =<M_{i+1},F(K_{i+2}), K_i>$ αποκαλύπτει το K_i και δίνει τη δυνατότητα στον παραλήπτη πρώτα να επαληθεύσει ότι το K_i είναι σωστό (αφού η $F(K_i)$ είναι ίση με τη δέσμευση κλειδιών που είχε σταλεί στο πακέτο P_{i-1} , και κατόπιν να υπολογίσει την τιμή $K_i'=F'(K_i)$ και να ελέγξει έτσι την αυθεντικότητα του πακέτου P_i επαληθεύοντας τον MAC του.

Μετά που ο αποστολέας έχει αυθεντικοποιήσει το P_i, η δέσμευση $F(K_{i+1})$ αυθεντικοποιείται επίσης. Ο παραλήπτης επαναλαμβάνει τη διαδικασία για να αυθεντικοποιήσει το P_{i+1} μετά που λαμβάνει το P_{i+2}.

Ο βασικός αλγόριθμος μπορεί να ανατραπεί εάν κάποιος κλέψει το πακέτο P_{i+1} πριν ο παραλήπτης λάβει το P_i, γιατί τότε ο εισβολέας θα γνωρίζει το κρυφό κλειδί K_i το οποίο χρησιμοποιείται για τον υπολογισμό του MAC που αντιστοιχεί στο πακέτο P_i και να αντικαταστήσει όλα τα επόμενα πακέτα. Για να διορθώσουμε αυτό το πρόβλημα, ο παραλήπτης ελέγχει την ακόλουθη συνθήκη ασφαλείας σε κάθε πακέτο που λαμβάνει και αγνοεί τα πακέτα που δεν την ικανοποιούν.

Συνθήκη ασφαλείας

> Ένα πακέτο P_i είναι ασφαλές εάν ο παραλήπτης μπορεί χωρίς καμιά αμφιβολία να αποφασίσει, βασιζόμενος στον αρχικό συγχρονισμό και στο σφάλμα δ_t, αν ο αποστολέας δεν έχει στείλει το αντίστοιχο πακέτο αποκάλυψης του κλειδιού P_j.

Ο αλγόριθμος είναι ασφαλής όσο ισχύει η συνθήκη ασφαλείας. Θα πρέπει να επισημάνουμε ότι δεν έχει γίνει κάποια υπόθεση για την καθυστέρηση του δικτύου.

Προκειμένου να επαληθεύσει ο παραλήπτης την συνθήκη ασφαλείας πρέπει να γνωρίζει το ακριβές σχεδιάγραμμα αποστολής των πακέτων. Ο πιο εύκολος τρόπος να γίνει αυτό είναι χρησιμοποιώντας ένα σταθερό ρυθμό αποστολής των πακέτων. Έτσι ο χρόνος αποστολής του πακέτου P_i είναι $T_i = T_0 + i/r$ όπου T_i είναι ο χρόνος σύμφωνα με το ρολόι του αποστολέα και r είναι ο ρυθμός αποστολής των πακέτων (αριθμός πακέτων που στέλνονται ανά δευτερόλεπτο). Σε αυτήν την περίπτωση η συνθήκη ασφαλείας μεταφράζεται στην σχέση $ArrT_i + \delta_t < T_{i+1}$, όπου $ArrT_i$ συμβολίζει τον χρόνο άφιξης του πακέτου P_i σύμφωνα με το συγχρονισμένο ρολόι του παραλήπτη. Το βασικό πρόβλημα είναι ότι προκειμένου να επαληθευτεί η συνθήκη ασφαλείας ο ρυθμός αποστολής των πακέτων πρέπει να είναι μικρότερος από την καθυστέρηση του δικτύου από τον αποστολέα στον παραλήπτη. Αυτός είναι ένας σημαντικός περιορισμός για την απόδοση της μετάδοσης. Επί προσθέτως, ο βασικός αλγόριθμος δεν μπορεί να αντεπεξέλθει σε ενδεχόμενη απώλεια των πακέτων. Πιο συγκεκριμένα όταν κάποιο πακέτο χαθεί κανένα μεταγενέστερο πακέτο δεν μπορεί να αυθεντικοποιηθεί. Προκειμένου να καλυπτούν οι παραπάνω ελλείψεις ο βασικός αλγόριθμος έχει επεκταθεί και αυτές ακριβώς οι αλλαγές παρουσιάζονται στην συνέχεια.

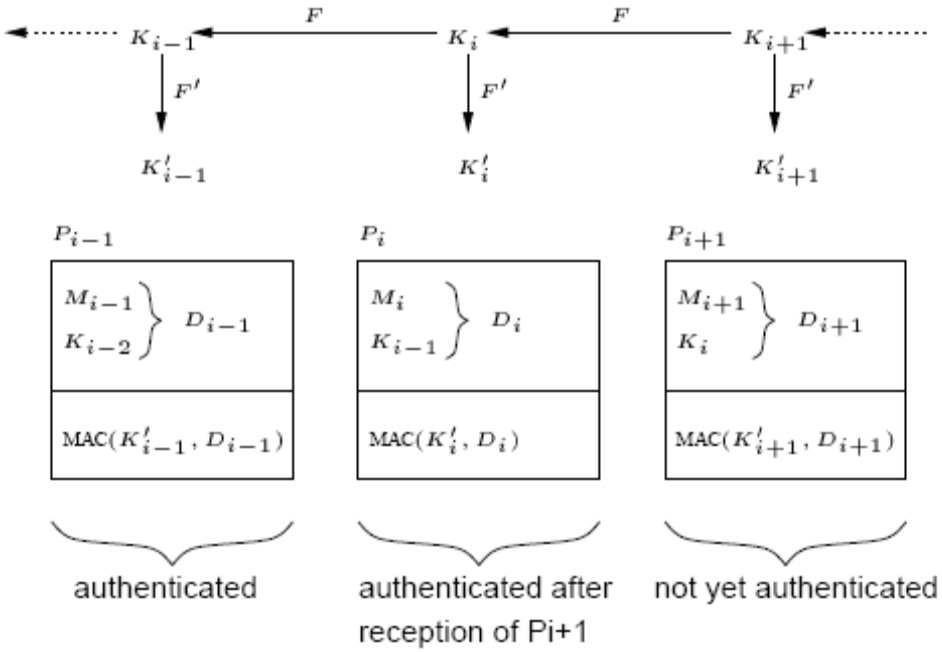
3.2.4 Αλγόριθμος 2: Ανάκτηση των κλειδιών από ενδεχόμενη απώλεια των πακέτων

Η βασική επέκταση του βασικού αλγορίθμου είναι η δημιουργία μιας ακολουθίας κλειδιών { K_i } μέσω διαδοχικών εφαρμογών ψευδοτυχαίων συναρτήσεων. Συμβολίζουμε *u* συνεχόμενες εφαρμογές της ψευδοτυχαίας συνάρτησης *F* ως $F^u(x) = F^{u-1}(F(x))$. Για λόγους ευκολίας θεωρούμε ότι $F^0(x) = x$. Ο αποστολέας επιλέγει ένα τυχαίο κλειδί K_n και υπολογίζει με τη βοήθεια αυτού μια ακολουθία *n* κλειδιών όπου $K_0=F^n(K_n)$, και $K_i=F^{n-i}(K_n)$. Καλούμε αυτήν την ακολουθία *αλυσίδα κλειδιών*. Κάθε ένα κλειδί φαίνεται ψευδοτυχαίο σε έναν εισβολέα. Πιο συγκεκριμένα δοθέντος του K_i ο εισβολέας δεν μπορεί να αντιστρέψει την *F* και να υπολογίσει οποιοδήποτε K_j όπου $j>i$. Από την άλλη, ο παραλήπτης μπορεί να υπολογίσει κάθε K_j από κάποιο K_i που έλαβε, όπου $j < i$,αφού $K_j = F^{i-j}(K_i)$. Επομένως εάν ένας παραλήπτης έλαβε το πακέτο P_i, κάθε επόμενο πακέτο που λαμβάνει θα του δίνει την ικανότητα να υπολογίζει το K_i και το $K_i' = F'(K_i)$ και να υπολογίζει έτσι την αυθεντικότητα του P_i. Με τον τρόπο αυτό μπορούμε να ανακτήσουμε τα κλειδιά ανεξαρτήτως από πόσα κλειδιά θα χαθούν.

Ομοίως, η απόρριψη των μη ασφαλών πακέτων (τα πακέτα τα οποία δεν ικανοποιούν την συνθήκη ασφαλείας) δεν προκαλεί κανένα πρόβλημα στην αυθεντικοποίηση των μεταγενέστερων από αυτά πακέτα.

Στον βασικό αλγόριθμο, ένας εισβολέας θα μπορούσε να προσπαθήσει να αιχμαλωτίσει δύο συνεχόμενα πακέτα πριν ο παραλήπτης να λάβει το πρώτο από αυτά και μετά να αντικαταστήσει όλα τα επόμενα πακέτα. Παρ' όλο που η συνθήκη ασφαλείας το αποτρέπει αυτό το αποτρέπει και η αλυσίδα των κλειδιών, αφού η αρχική δέσμευση αφορά όλη την αλυσίδα οπότε είναι υπολογιστικά αδύνατο για τον επιτιθέμενο να αντιστρέψει και να βρει κάποιες συγκρούσεις στην ψευδοτυχαία συνάρτηση.

Ένα ακόμη πλεονέκτημα είναι ότι πλέον η δέσμευση των κλειδιών δεν είναι αναγκαίο να ενσωματώνεται στα πακέτα. Αφού είναι πάρα πολύ δύσκολο να αντιστρέψει κανείς μια ψευδοτυχαία συνάρτηση, κάθε κλειδί που ανήκει στην αλυσίδα αποτελεί μια δέσμευση για ολόκληρη την αλυσίδα. Στο σχήμα 3-2 αναπαριστούνται οι βασικές ιδέες του δεύτερου αλγορίθμου.



Σχήμα 3-2

Η δομή των πακέτων παραμένει όπως στον βασικό αλγόριθμο, εκτός του ότι η δέσμευση $F(K_{i-1})$ παραλείπεται και τα κλειδιά σχηματίζουν μια μονόδρομη αλυσίδα.

3.2.5 Αλγόριθμος 3: Επιτυγχάνοντας γρήγορους ρυθμούς μεταφοράς

Όπως αναφέραμε νωρίτερα, ο παραλήπτης χρειάζεται να σιγουρευτεί ότι έλαβε το πακέτο P_i πριν να στείλει ο αποστολέας το πακέτο αποκάλυψης του αντίστοιχου κλειδιού P_{i+1}. Αυτή η συνθήκη περιορίζει αρκετά το ρυθμό μετάδοσης των δύο προηγούμενων αλγορίθμων αφού το P_{i+1} μπορεί να σταλεί μόνο αφού κάθε αποστολέας λάβει το P_i.

Λύνουμε αυτό το πρόβλημα αποκαλύπτοντας το κλειδί K_i που αντιστοιχεί στο πακέτο δεδομένων P_i μετά από d πακέτα, δηλαδή σε ένα επόμενο πακέτο P_{i+d} αντί στο αμέσως επόμενο πακέτο. Η παράμετρος d λέγεται παράμετρος καθυστέρησης αποκάλυψης των κλειδιών και ορίζεται από τον αποστολέα. Η παράμετρος καθυστέρησης αποκάλυψης των κλειδιών ανακοινώνεται από τον αποστολέα στους παραλήπτες κατά τη διάρκεια της αρχικοποίησης, ενώ καθορίζει την τιμή της βασιζόμενος στον ρυθμό αποστολής των πακέτων r, το μέγιστο σφάλμα συγχρονισμού δ_{tMax}, και την μέγιστη καθυστέρηση δικτύου d_{NMax}. Ορίζοντας ως $d = \lceil (\delta_{tMax} + d_{NMax})r \rceil$ ο παραλήπτης δύναται να επαληθεύσει επιτυχώς την συνθήκη ασφαλείας ακόμη και στην περίπτωση της μέγιστης επιτρεπτής καθυστέρησης δικτύου και του μέγιστου σφάλματος συγχρονισμού. Η επιλογή του δ_{tMax} και του d_{NMax} αποτελεί ένα δίλημμα. Μεγάλες τιμές καθυστέρησης δικτύου προκαλούν μεγάλο d το οποίο έχει ως αποτέλεσμα μεγάλη καθυστέρηση μέχρι την αυθεντικοποίηση των πακέτων. Από την άλλη, μικρές μέγιστες καθυστερήσεις θα προκαλέσουν την απόρριψη πολλών πακέτων από

την συνθήκη ασφαλείας για παραλήπτες με αργές συνδέσεις δικτύου. Όμως, τα πακέτα δεδομένων που μεταφέρουν πολυμέσα (π.χ. εικόνα και ήχο) αχρηστεύονται αν η θέση στην οποία αντιστοιχούν έχει ήδη παιχτεί (αυτό αποτελεί αιτία χαμένων σκηνών κτλ). Σε αυτήν την περίπτωση, η απόρριψη των μη ασφαλών πακέτων μπορεί να μην χρειάζεται αφού τα πακέτα πιθανόν να έχουν γίνει άχρηστα. Τονίζουμε ότι η επιλογή του d δεν επηρεάζει την ασφάλεια του αλγορίθμου, αλλά μόνο την χρησιμότητα του.

Στην περίπτωση του σταθερού ρυθμού πακέτων μετάδοσης, η συνθήκη ασφαλείας, είναι εύκολο να καθοριστεί. Υποθέτουμε ότι ο χρόνος αποστολής του πρώτου πακέτου είναι T_0 και ο χρόνος αποστολής του πακέτου P_i είναι $T_i = T_0 + i/r$. Για να επαληθεύσουμε την συνθήκη ασφαλείας για ένα εισερχόμενο πακέτο, ο παραλήπτης ελέγχει την ανισότητα $ArrT_i + \delta_t < T_{i+d}$ όπου $ArrT_i$ είναι ο χρόνος άφιξης του πακέτου P_i στον παραλήπτη.

3.2.6 Αλγόριθμος 4: Αντιμετωπίζοντας μεταβλητούς ρυθμούς αποστολής δεδομένων

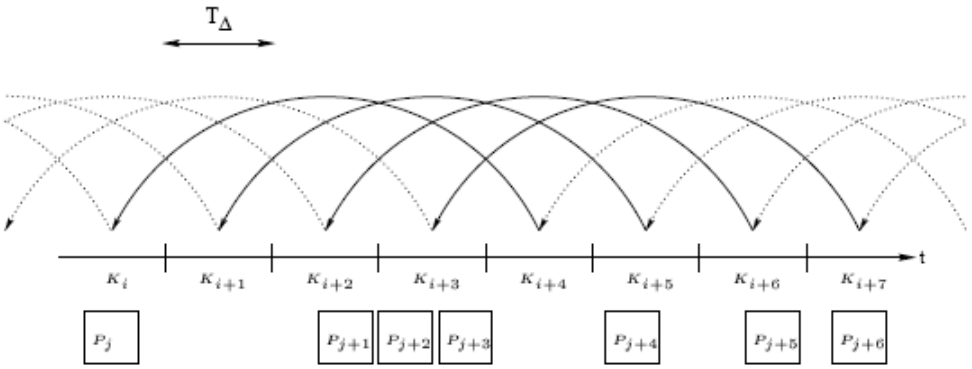
Οι προηγούμενοι αλγόριθμοι χρησιμοποιούσαν ένα σταθερό και προβλεπόμενο σχεδιάγραμμα αποστολής πακέτων, όπου ο κάθε παραλήπτης γνωρίζει τον ακριβή χρόνο αποστολής των πακέτων. Αφού αυτό περιορίζει σοβαρά την ευελιξία των αποστολέων, σχεδιάζουμε ένα αλγόριθμο που επιτρέπει στους αποστολείς να στείλουν δεδομένα σε δυναμικά μεταβαλλόμενους ρυθμούς αποστολής, χωρίς την απαίτηση ο κάθε αποστολέας να γνωρίζει τον ακριβή χρόνο αποστολής του κάθε πακέτου. Η λύση του προβλήματος, προκύπτει επιλέγοντας το κλειδί MAC και το αποκαλυφθέν κλειδί για κάθε πακέτο με βάση το χρονικό διάστημα που αυτό ανήκει και όχι σύμφωνα με την αρίθμηση του πακέτου. Ο αποστολέας χρησιμοποιεί το ίδιο κλειδί K_i για να υπολογίσει τη MAC για όλα τα πακέτα που στέλνονται στο ίδιο διάστημα i. Όλα τα πακέτα που έχουν σταλεί στο διάστημα i αποκαλύπτουν το κλειδί K_{i-d}. Στην φάση των αρχικοποιήσεων ο αποστολέας ανακοινώνει τις τιμές T_0 και T_Δ, όπου η πρώτη είναι ο χρόνος έναρξης του πρώτου διαστήματος και η τελευταία η διάρκεια κάθε διαστήματος. Ακόμη ανακοινώνεται η παράμετρος καθυστέρησης αποκάλυψης κλειδιών d (σε χρονικά διαστήματα). Οι ανακοινώσεις αυτές υπογράφονται ψηφιακά από τον αποστολέα. Η αρίθμηση των χρονικών διαστημάτων για κάθε χρονική στιγμή t ορίζεται ως $i = \lfloor (t - T_0)/T_\Delta \rfloor$. Κάθε κλειδί K_i αντιστοιχεί σε ένα χρονικό διάστημα i. Τα κλειδιά σχηματίζουν αλυσίδα σύμφωνα με τον αλγόριθμο 2. Ο αποστολέας χρησιμοποιεί το ίδιο κλειδί $K_i' = F'(K_i)$ για να υπολογίσει τον MAC (=Hash(Message+secret key)) για το κάθε πακέτο το οποίο στέλνεται στο χρονικό διάστημα i . Επίσης κάθε πακέτο i μεταφέρει την αρίθμηση του χρονικού διαστήματος i και αποκαλύπτει το κλειδί ενός προηγούμενου K_{i-d}. Αναφερόμαστε στο d ως καθυστέρηση

αποκάλυψης κλειδιών. Η δομή του πακέτου P_j είναι $P_j = < M_j, i, K_{i-d}, MAC(K_i', M_j) >$. Στο σχήμα 3-3 φαίνεται ένα παράδειγμα αυτού του αλγορίθμου, όπου $d=4$. Σε αυτόν τον αλγόριθμο, ο παραλήπτης επαληθεύει την συνθήκη ασφαλείας ως εξής: Κάθε παραλήπτης γνωρίζει τις τιμές T_0 , T_Δ και δ_t ,όπου δ_t είναι η τιμή που λαμβάνεται από το αρχικό πρωτόκολλο συγχρονισμού. Έστω ότι ο αποστολέας λαμβάνει το πακέτο P_j στον τοπικό χρόνο t_j και ότι το πακέτο εστάλη στο διάστημα i. Ο αποστολέας μπορεί το πολύ να είναι στο διάστημα $i' = \lfloor (t_j + \delta_t - T_0)/T_\Delta \rfloor$. Η συνθήκη ασφαλείας σε αυτήν την περίπτωση είναι απλά $i+d > i'$, η οποία διαβεβαιώνει ότι κανένα πακέτο το οποίο αποκαλύπτει την τιμή των κλειδιών δεν έχει σταλεί πριν να περάσουν d χρονικές μονάδες. Το σχήμα 3-4 απεικονίζει την επαλήθευση της συνθήκης ασφαλείας.

Μένει να περιγράψουμε πως οι τιμές T_Δ και d επιλέγονται. Τονίζουμε ότι αυτές οι τιμές δεν επηρεάζουν την ασφάλεια του αλγορίθμου, αλλά μόνο την χρηστικότητα του. Πριν ο αποστολέας επιλέξει τιμές για τα T_Δ και d πρέπει να καθορίσει την μέγιστη ανεκτή αβεβαιότητα συγχρονισμού και την μέγιστη ανεκτή καθυστέρηση δικτύου d_{NMax}. Ο αποστολέας ορίζει $\Delta_{Max} = d_{NMax} + \delta_{tMax}$. Οι επιλογές του αποστολέα για τα T_Δ και Δ_{Max} αποτελούν δίλημμα. Πιο αναλυτικά μια μεγάλη τιμή για το Δ_{Max} θα επέτρεπε σε παραλήπτες με αργή σύνδεση να επαληθεύσουν την συνθήκη ασφαλείας, αλλά με μεγάλες καθυστερήσεις για την αυθεντικοποίηση των πακέτων. Αντιστρόφως, ένα μικρό Δ_{Max} θα προκαλέσει στους παραλήπτες με αργή σύνδεση να επαληθεύσουν την συνθήκη ασφαλείας, αλλά με μεγάλες καθυστερήσεις για την αυθεντικοποίηση των πακέτων. Ομοίως, ένα μικρό Δ_{Max} θα προκαλέσει στους παραλήπτες με αργή σύνδεση να απορρίπτουν πολλά πακέτα γιατί δεν θα ικανοποιείται η συνθήκη ασφαλείας. Το δεύτερο δίλημμα είναι ότι μεγάλες τιμές της διάρκειας T_Δ συμφέρει όσον αφορά το υπολογιστικό κόστος και το κόστος αποθήκευσης της αλυσίδας των κλειδιών, αλλά ένα μικρό T_Δ προσεγγίζει καλύτερα το ζητούμενο Δ_{Max}.

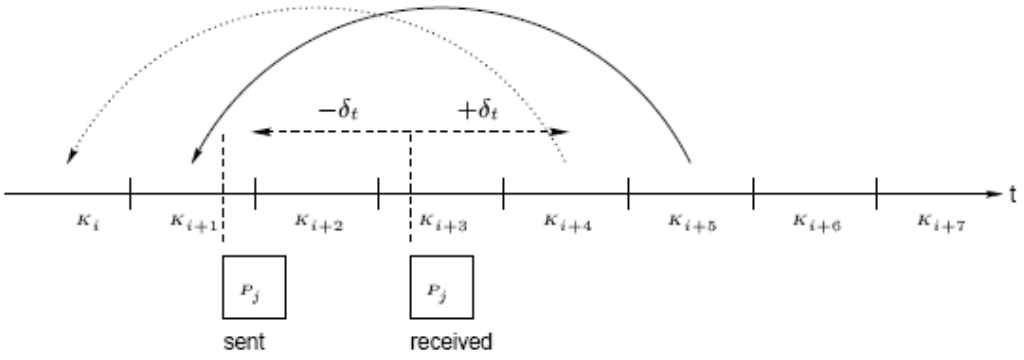
Μετά που θα καθοριστούν τα d_{NMax}, δ_{tMax} και T_Δ , η καθυστέρηση αποκάλυψης ορίζεται $d = \lceil (\delta_{tMax} + d_{NMax}) / T_\Delta \rceil$.

Αυτός ο αλγόριθμος παρουσιάζει πολυάριθμα πλεονεκτήματα. Πρώτα ο αποστολέας μπορεί να προβλέψει πόσο διαρκεί μια προϋπολογισμένη αλυσίδα κλειδιών, αφού ο αριθμός των κλειδιών εξαρτάται μόνο από τον χρόνο και όχι από τον αριθμό των πακέτων που στέλνονται. Δεύτερον, ο παραλήπτης μπορεί εύκολα να επαληθεύσει την συνθήκη ασφαλείας και ο αποστολέας δεν χρειάζεται να στείλει τα πακέτα σε συγκεκριμένα διαστήματα. Τρίτον καινούριοι παραλήπτες μπορούν εύκολα να συμμετάσχουν στην ομάδα των παραληπτών όποτε αυτοί το θελήσουν. Μια νέα ομάδα μελών αρκεί να συγχρονίσει το ρόλοι της με του αποστολέα, να λάβει τις παραμέτρους των διαστημάτων και την αρχική δέσμευση για το κλειδί από τον αποστολέα προκειμένου να αρχίσει να λαμβάνει δεδομένα.



Σχήμα 3-3

Αλγόριθμος 4.Το κλειδί MAC και το αποκαλυφθέν κλειδί εξαρτώνται μόνο από τα χρονικά διαστήματα. Το κλειδί που χρησιμοποιείται για την αυθεντικοποίηση του P_j *είναι το* K_i το οποίο αποκαλύπτεται από πακέτα που στέλνονται στο διάστημα *i+4*. Σε αυτήν την περίπτωση το πακέτο P_{j+4} αποκαλύπτει το κλειδί K_{i+1} που επιτρέπει στους παραλήπτες να υπολογίσουν το K_i και να αυθεντικοποιήσουν το πακέτο P_j . Ωστόσο οι *MACs* των πακέτων P_{j+2} *και* P_{j+3} παράγονται από το ίδιο κλειδί K_{i+3} ΄, γιατί στάλθηκαν στο ίδιο διάστημα.



Σχήμα 3-4

Το πακέτο P_j στέλνεται στο διάστημα όπου το κλειδί K_{i+1} είναι ενεργό. Εδώ d=4. Ο παραλήπτης λαμβάνει το πακέτο όταν ο αποστολέας είναι στο διάστημα *i+3*, αλλά σύμφωνα με το δ_t ο αποστολέας μπορεί να είναι ήδη στο *i+4,* όπου αποκαλύπτει το K_i. Αυτό δεν είναι πρόβλημα για το τρέχων πακέτο, έτσι το κλειδί K_{i+1} δεν αποκαλύπτεται ακόμη, αφού η συνθήκη ασφαλείας ικανοποιείται και το πακέτο P_i είναι ασφαλής.

3.2.7 Αλγόριθμος 5: Εξυπηρετώντας ένα μεγάλο φάσμα παραληπτών

Για τους προηγούμενους αλγόριθμους, δείξαμε ότι υπήρχαν κάποια διλήμματα στις επιλογές της περιόδου αποκάλυψης των κλειδιών. Εάν η περίοδος είναι μικρή, τα πακέτα μπορούν να αυθεντικοποιηθούν γρήγορα, αλλά αν ο χρόνος της διάδοσης του πακέτου είναι μεγάλος η συνθήκη ασφαλείας θα προκαλεί την απόρριψη πολλών πακέτων για τους απομακρυσμένους παραλήπτες. Αντιστρόφως μια μεγάλη περίοδος αποκάλυψης θα είναι κατάλληλη για απομακρυσμένους παραλήπτες, αλλά η καθυστέρηση αυθεντικοποίησης ίσως είναι μη αποδεκτή για παραλήπτες με γρήγορες συνδέσεις. Αφού ο αλγόριθμος θα πρέπει να εξυπηρετεί ένα μεγάλο αριθμό από παραλήπτες και περιμένουμε οι παραλήπτες να έχουν ποικίλλων ειδών συνδέσεις , θα πρέπει να βρούμε μια λύση. Η προσέγγιση του *TESLA* είναι η χρήση πολλαπλών αλυσίδων αυθεντικοποίησης με διαφορετικές καθυστερήσεις αποκάλυψης ταυτοχρόνως. Κάθε παραλήπτης μπορεί μετά να χρησιμοποιήσει την αλυσίδα με την μικρότερη καθυστέρηση, που επαρκεί για να αποτρέψει λανθασμένες απορρίψεις πακέτων από λάθος εκτίμηση της συνθήκης ασφαλείας.

Ο παραλήπτης επαληθεύει μια συνθήκη ασφαλείας για κάθε αλυσίδα αυθεντικοποίησης C_i και απορρίπτει τα πακέτα εάν καμία από τις συνθήκες δεν ικανοποιείται. Έστω ότι ο αποστολέας χρησιμοποιεί n αλυσίδες αυθεντικοποίησης, όπου η πρώτη έχει τη μικρότερη καθυστέρηση αποκάλυψης και η n-στη την μεγαλύτερη. Επιπλέον έστω ότι για τα εισερχόμενα πακέτα P_j ,η συνθήκη ασφαλείας για τις αλυσίδες C_u $(u<m)$δεν ικανοποιούνται ενώ για τις αλυσίδες C_m ικανοποιείται. Σε αυτήν την περίπτωση ,όσο φτάνουν τα πακέτα αποκάλυψης των κλειδιών για τις αλυσίδες C_u $(u<m)$, η βεβαιότητα των παραληπτών για την αυθεντικότητα των πακέτων P_j αυξάνει. Μόλις λάβουν το πακέτο αποκάλυψης των κλειδιών για μια από τις αλυσίδες C_u ο αποστολέας σιγουρεύεται εντελώς για την αυθεντικότητα του πακέτου P_j .

3.2.7 Ανάλυση του αρχικού πρωτοκόλλου συγχρονισμού

Το *TESLA* βασίζεται στον μη αυστηρό συγχρονισμό των ρολογιών μεταξύ αποστολέα και αποδεκτών. Ονομάζουμε τον συγχρονισμό μη αυστηρό, γιατί το σφάλμα συγχρονισμού μπορεί να είναι μεγάλο. Οι μόνες απαιτήσεις που υπάρχουν είναι ότι οι αποδέκτες γνωρίζουν ένα άνω όριο δ_t του μέγιστου σφάλματος συγχρονισμού.

Οποιοδήποτε πρωτόκολλο συγχρονισμού μπορεί να χρησιμοποιηθεί, αρκεί να είναι ανθεκτικό απέναντι σε επιθέσεις από το μοντέλο - εισβολέα.

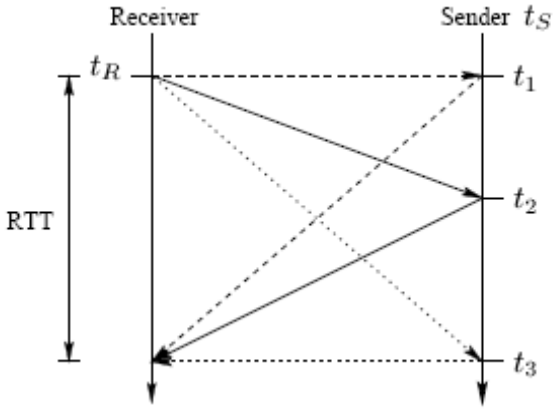
Ως απόδειξη του πάνω συλλογισμού το *TESLA*, παρουσιάζει ένα απλό πρωτόκολλο συγχρονισμού το οποίο ικανοποιεί τις προαναφερθείσες προϋποθέσεις. Το βασικό πρωτόκολλο έχει ως εξής:

R → S: nonce

S → R: {Sender time t_s, nonce, Interval Rate, Interval Id, Interval start time, Interval Key, Disclosure Lag} $_{Ks}^{-1}$

Ο αποστολέας χρησιμοποιεί μια nonce στο πρώτο του πακέτο για να εμποδίσει μια επίθεση η οποία περιλαμβάνει μια πρόσφατη υπογεγραμμένη συγχρονισμένη απάντηση. Παράλληλα με τον τρέχον χρόνο t_s του αποστολέα, ο αποστολέας στέλνει και όλες τις πληροφορίες που είναι απαραίτητες για να ορίσει τα χρονικά διαστήματα και την δέσμευση στην ενεργή αλυσίδα των κλειδιών. Η καθυστέρηση της αποκάλυψης των κλειδιών ορίζει μετά από πόσα διαστήματα θα αποκαλύπτονται τα κλειδιά. Τέλος, το πακέτο υπογράφεται από τον αποστολέα σύμφωνα με ένα συνηθισμένο αλγόριθμο υπογραφής (π.χ. *RSA*).

Οι παραλήπτες ενδιαφέρονται μόνο για τον μέγιστο δυνατό χρόνο που μπορεί να έχει ο αποστολέας. Αυτό απλοποιεί τους υπολογισμούς. Στο *σχήμα 3-5* παριστάνεται το χρονοδιάγραμμα του συγχρονισμού. Ο παραλήπτης θέτει $\Delta_t = t_s - t_r$ και υπολογίζει τον τελευταίο δυνατό χρόνο του αποστολέα t_s' ως εξής : $t_s' = t_r' + \Delta_t$, όπου t_r' είναι ο τρέχων χρόνος του παραλήπτη και t_s' ο εκτιμούμενος χρόνος του αποστολέα. Σε ιδανικές καταστάσεις, το αρχικό πακέτο φτάνει στον παραλήπτη χωρίς καθόλου καθυστέρηση σε χρόνο t_1 όπως συμβολίζεται στο σχήμα. Η μέγιστη απώλεια χρόνου είναι δ_t =*RTT (round trip time)*.



Σχήμα 3-5*:Ο παραλήπτης συγχρονίζει την ώρα του με τον αποστολέα.*

Η ευελιξία είναι μέγιστης σημασίας για ένα αρκετά μεγάλο σύστημα. Εάν κάθε παραλήπτης πρέπει να συγχρονίζει την ώρα του με του αποστολέα τότε ο αποστολέας ίσως αποτελεί στένωση για το σύστημα. Μια καλύτερη προσέγγιση θα χρησιμοποιούσε κατανεμημένους και

ασφαλής εξυπηρετητές χρόνους *(time servers)*. Αρχικά ο αποστολέας συγχρονίζει την ώρα του με τον εξυπηρετητή χρόνου και υπολογίζει το μέγιστο σφάλμα συγχρονισμού $\delta_t(S)$. Ο αποστολέας θα μπορούσε να μεταδίδει σε όλους την αρίθμηση των χρονικών διαστημάτων, μαζί με το $\delta_t(S)$ και την τρέχουσα σφραγίδα χρόνου *(timestamp)*, υπογεγραμμένα ψηφιακά για να εξασφαλιστεί η αυθεντικότητα. Οι παραλήπτες μπορούν να συγχρονίσουν ανεξάρτητα το χρόνο τους με τον εξυπηρετητή συγχρονισμού και να υπολογίσουν ο καθένας το μέγιστο σφάλμα συγχρονισμού δ_t.Τελικά οι παραλήπτες προσθέτουν όλα τα δ_t για να υπολογίσουν το τελικό σφάλμα συγχρονισμού δ_t. Πηγαίνοντας ένα βήμα πιο πέρα, το *TESLA* θα μπορούσε να διαθέτει μια ιεράρχηση στους εξυπηρετητές συγχρονισμού (όπου θα διαδίδονται μόνο τα μέγιστα σφάλματα διάδοσης) και να συγχρονίζονται όλοι οι εξυπηρετητές μέσω κάποιου δορυφορικού συστήματος, για παράδειγμα *GPS*.

3.2.8 Συνδυασμός του TESLA με κέντρα ελέγχου ομάδων πολυεκπομπής δεδομένων

Το γενικό μοντέλο πολυεκπομπής δεδομένων *IP* υποθέτει ότι οποιοσδήποτε υπολογιστής – χρήστης μπορεί να συμμετάσχει στην ομάδα λήψης δεδομένων, να λάβει όλα τα δεδομένα που προορίζονται για την ομάδα αλλά και να στείλει δεδομένα σε αυτήν. Για να γίνει μέλος της ομάδας κάποιος παραλήπτης θα πρέπει να εκδηλώσει το ενδιαφέρον του σε ένα τοπικό δρομολογητή ο οποίος στη συνέχεια θα προωθεί τα πακέτα σε αυτόν. Κάθε νέο μέλος πρέπει να επικοινωνήσει με τον κεντρικό εξυπηρετητή ή τον υπεύθυνο της ομάδας για να διαπραγματευτεί δικαιώματα πρόσβασης και κλειδιά συμμετοχής στην ομάδα. Αυτό το μοντέλο υποστηρίζεται από τις ομάδες *Secure Multicast Users Group* (SMUG) [29] και το *TESLA* το ασπάζεται, πράγμα που σημαίνει ότι κάθε παραλήπτης θα πρέπει να πραγματοποιήσει μια αρχική εγγραφή στον κεντρικό εξυπηρετητή ή στον αποστολέα προκειμένου να λάβει τις αρχικές πληροφορίες συγχρονισμού.

Ακολουθεί ένα παράδειγμα αυτής της τακτικής. Έστω ότι τόσο οι αποστολείς αλλά και οι παραλήπτες συγχρονίζονται με κάποιους εξυπηρετητές συγχρονισμού οι οποίο υπάρχουν διασκορπισμένοι στο δίκτυο. Μετά το συγχρονισμό κάθε οντότητα E γνωρίζει το $\delta_t(E)$.Ο αποστολέας S μεταδίδει σε όλο το σύστημα περιοδικά ένα υπογεγραμμένο μήνυμα το οποίο περιέχει $\delta_t(S)$, μαζί με το διάστημα και την δέσμευση στην αλυσίδα των κλειδιών. Άρα ένας νέος παραλήπτης R χρειάζεται μόνο να περιμένει για την μετάδοση αυτών των πληροφοριών για να υπολογίσει το σφάλμα συγχρονισμού μεταξύ του εαυτού του και του αποστολέα σύμφωνα με τον τύπο : $\delta_t = \delta_t(S) + \delta_t(R)$. Βασιζόμενος στο δ_t ο παραλήπτης αποφασίζει ποια αλυσίδα αυθεντικοποίησης θα διαλέξει επιλέγοντας αυτήν με τη μικρότερη καθυστέρηση. Με αυτόν τον τρόπο ο παραλήπτης δεν είναι αναγκασμένος να στείλει άλλα μηνύματα στον

αποστολέα, με την προϋπόθεση ότι αποστολέας και παραλήπτης διαθέτουν μια μέθοδο να συγχρονιστούν και ο παραλήπτης γνωρίζει το άνω όριο του σφάλματος συγχρονισμού δ_t.

3.2.9 Αντιμετωπίζοντας κύλιση χρόνου στο ρολόι του αποστολέα

Στους παραπάνω αλγόριθμους έχει υποτεθεί ότι δεν υπάρχει κύλιση χρόνου μεταξύ του αποστολέα και του παραλήπτη. Παρ' όλα αυτά όμως στην πραγματικότητα τα ρολόγια μπορούν να μη μετράνε σωστά (για παράδειγμα όταν το σύστημα είναι υπερφορτωμένο). Επίσης ένας εισβολέας ίσως να καταφέρει να αλλάξει τον χρόνο του θύματος. Μια λύση σε αυτά τα προβλήματα είναι ότι ο παραλήπτης πάντα συμβουλεύεται το εσωτερικό ρολόι του συστήματος του, το οποίο κυλάει ελάχιστα και είναι πολύ δύσκολο να το ενοχλήσει κάποιος εισβολέας. Περαιτέρω, η μεγαλύτερη αλυσίδα αυθεντικοποίησης του αλγόριθμου 5 ανέχεται καθυστέρηση αυθεντικοποίησης της τάξεως των δέκατων δευτερολέπτων και μας δίνει μεγάλα περιθώρια ασφάλειας. Είναι λογικό να υποθέσει κανείς ότι το ρολόι του συστήματος δεν κυλάει κατά δέκατα του δευτερολέπτου σε μια περίοδο. Εν τέλει ο παραλήπτης μπορεί και να επανασυγχρονίζεται περιοδικά, εάν το ρόλοι του συστήματος εμφανίζεται να κυλάει σημαντικά.

4

Ανάλυση Απαιτήσεων Συστήματος

Θα ακολουθήσει η περιγραφή της αρχιτεκτονικής του συστήματος *TESLA*. Η υλοποίηση του συστήματος έγινε στη γλώσσα *JAVA*. Ακόμη θα γίνει η ανάλυση απαιτήσεων για τις λειτουργίες του.

4.1 Αρχιτεκτονική

Το σύστημα *TESLA* αποτελείται από δυο βασικά υποσυστήματα – οντότητες οι οποίες αλληλεπιδρούν μεταξύ τους ανταλλάσσοντας μεταξύ τους μηνύματα στη μορφή των πακέτων δεδομένων. Τα δυο υποσυστήματα είναι ο αποστολέας *(sender)* και ο παραλήπτης *(receiver)*. Αυτά προσομοιώνονται στη γλώσσα *JAVA* σύμφωνα με την αρχιτεκτονική *client/server* 2 επιπέδων. Ακολουθεί μια σύντομη παρουσίαση των βασικών εννοιών της αρχιτεκτονικής *client/server*.

4.1.1 Αρχιτεκτονική client / server

Έστω ότι έχουμε δυο διεργασίες οι οποίες εκτελούνται σε δυο μηχανήματα τα οποία συνδέονται μέσω δικτύου, τοπικού *(LAN)* ή ακόμη και δικτύου ευρείας περιοχής *(WAN)*.Οι δυο διεργασίες ονομάζονται *client* και *server*.Οι διεργασίες θα μπορούσαν να εκτελούνται ακόμη και στο ίδιο μηχάνημα. Ακόμη θεωρούμε ότι οι δυο διεργασίες προσπαθούν να επικοινωνήσουν η μια με την άλλη και να ανταλλάξουν πληροφορίες. Προκειμένου να το πετύχουν θα πρέπει να συμφωνήσουν στον τρόπο με τον οποίο θα επιτευχθεί αυτή η επικοινωνία. Στη συνέχεια εξετάζουμε την κάθε διεργασία χωριστά και παρουσιάζουμε το ρόλο της κάθε μιας.

4.1.1.1 Η διεργασία client

Ο πελάτης *(client)* είναι μια διεργασία (πρόγραμμα), η οποία τρέχει στο μηχάνημα του χρήστη, και στέλνει ένα μήνυμα στη διεργασία του εξυπηρετητή *(server)*, απαιτώντας από τον εξυπηρετητή να εκτελέσει μια εφαρμογή ή υπηρεσία *(task or service)* για αυτόν. Τα προγράμματα – πελάτες συνήθως διαχειρίζονται την γραφική διεπαφή χρήστη του συστήματος, τα έγκυρα δεδομένα που εισάγονται από τον χρήστη και μεταδίδουν τις αιτήσεις *(requests)* στα προγράμματα εξυπηρετητές. Οι εφαρμογές – πελάτες σε ένα υπολογιστή αποτελούν το μέτωπο της εφαρμογής και είναι αυτό που ο χρήστης βλέπει και αλληλεπιδρά.

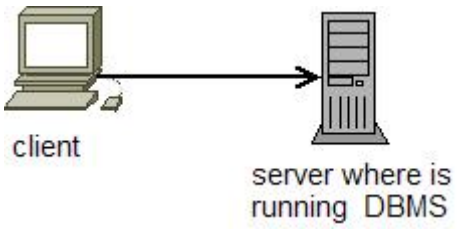
Ακόμα διαχειρίζονται τους πόρους του συστήματος το οποίο χειρίζεται ο χρήστης, όπως το πληκτρολόγιο, η οθόνη, η *CPU* και τα υπόλοιπα περιφερειακά. Ένα από τα στοιχεία κλειδιά των εφαρμογών πελάτη είναι η γραφική διεπαφή χρήστη. Συνήθως ένα μέρος του λειτουργικού συστήματος π.χ. ένας *window manager* ανιχνεύει τις κινήσεις χρήστη, διαχειρίζεται τα παράθυρα κατά την απεικόνιση και εμφανίζει τα δεδομένα στα παράθυρα.

4.1.1.2 Η διεργασία server

Μια διεργασία εξυπηρετητή *(server)* ανταποκρίνεται στις αιτήσεις του πελάτη *(client)* εκτελώντας τις εφαρμογές που εκείνος αιτείται. Γενικότερα τα προγράμματα – εξυπηρετητές δέχονται αιτήσεις από τα προγράμματα – πελάτες , ανασύρουν δεδομένα από τις βάσεις δεδομένων και ανανεώνουν αυτές, διαχειρίζονται την ακεραιότητα των δεδομένων και απαντούν στις αιτήσεις των πελατών. Οι διεργασίες εξυπηρετητή είναι δυνατό να εκτελούνται σε διαφορετικό υπολογιστή του δικτύου απ' ότι η διεργασία του πελάτη. Ο εξυπηρετητής συνήθως παρέχει υπηρεσίες διανομής των αρχείων του συστήματος *(FTP)*, εκτελεί κάποιες υπηρεσίες – εφαρμογές ή ακόμη σε άλλες περιπτώσεις μεταδίδει την αίτηση σε άλλο εξυπηρετητή καταλληλότερο για την εκτέλεση της εφαρμογής. Οι εξυπηρετητές συνήθως εκτελούν όλες τις απαραίτητες λειτουργίες, προκειμένου να ολοκληρωθούν οι αιτήσεις των πελατών του συστήματος οι οποίες όμως δεν φαίνονται και εκτελούνται στο παρασκήνιο και όχι στο προσκήνιο όπως είναι με τις διεργασίες – πελάτη. Κάποια είδη εξυπηρετητών είναι οι εξυπηρετητές αρχείων *(file servers)*, οι εξυπηρετητές εφαρμογών *(application servers)*, οι εξυπηρετητές ταχυδρομείου *(mail servers)*, οι εξυπηρετητές τερματικών *(terminal servers)*.

4.1.1.3 Η αρχιτεκτονική client / server 2 επιπέδων

Η αρχιτεκτονική *client / server* 2 επιπέδων υφίσταται όταν υπάρχει απευθείας επικοινωνία μεταξύ της διεργασίας – εξυπηρετητή και της διεργασίας – πελάτη χωρίς να μεσολαβεί κάποιος άλλος εξυπηρετητής. Αυτό σημαίνει ότι στο δίκτυο υπάρχουν δυο ειδών κόμβοι, οι εξυπηρετητές και οι πελάτες. Συνήθως χρησιμοποιείται σε περιβάλλοντα μικρού μεγέθους (αριθμός χρηστών μικρότερος από 50) και ο εξυπηρετητής είναι εξυπηρετητής βάσης δεδομένων. Η αρχιτεκτονική αυτού του είδους απεικονίζεται στο σχήμα 4.1.



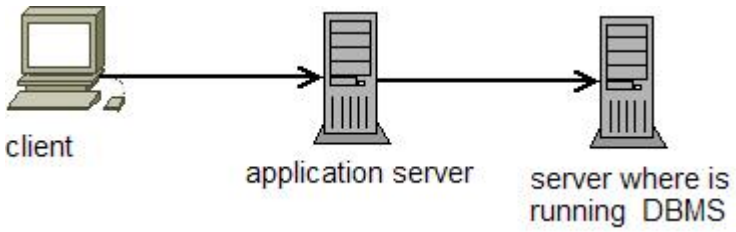
Σχήμα 4.1:Αρχιτεκτονική client/server 2 επιπέδων

Ένα συνηθισμένο πρόβλημα στην ανάπτυξη *client / server* εφαρμογών είναι το εξής: Έστω ότι δημιουργούμε ένα σύστημα client / server 2 επιπέδων σε ένα περιβάλλον μικρού μεγέθους

και στη συνέχεια προσπαθούμε να επεκτείνουμε αυτό το περιβάλλον προσθέτοντας περισσότερους χρήστες στον εξυπηρετητή. Η προσπάθεια αυτή συνήθως έχει ως αποτέλεσμα το σύστημα να καταλήγει αναποτελεσματικό, αφού ο εξυπηρετητής κατακλύζεται από τις αιτήσεις των πελατών και δεν προλαβαίνει να τους εξυπηρετήσει επιτυχώς. Για να επιτευχθεί η επέκταση του συστήματος σε εκατοντάδες ή ακόμη και σε χιλιάδες χρήστες, είναι αναγκαίο να χρησιμοποιηθεί η αρχιτεκτονική 3 επιπέδων η οποία και περιγράφεται στη συνέχεια.

4.1.1.4 Η αρχιτεκτονική client / server 3 επιπέδων

Η αρχιτεκτονική *client / server* 3 επιπέδων (σχήμα 4.2) εισάγει στο προηγούμενο μοντέλο ακόμη ένα εξυπηρετητή ή αλλιώς ένα πράκτορα *(agent),* ο οποίος μεσολαβεί μεταξύ των ήδη υπαρχόντων πελάτη και εξυπηρετητή. Αυτό σημαίνει ότι στο δίκτυο υπάρχουν τρία είδη κόμβων, οι πελάτες, οι εξυπηρετητές εφαρμογών (δηλαδή οι πράκτορες) οι οποίοι επεξεργάζονται τα δεδομένα για τους πελάτες και οι εξυπηρετητές βάσεων δεδομένων οι οποίοι αποθηκεύουν τα δεδομένα για τους εξυπηρετητές εφαρμογών. Ο χαρακτήρας του πράκτορα είναι πολυπρόσωπος. Μπορεί να παρέχει υπηρεσίες μετάφρασης *(translation services)*, υπηρεσίες καταμέτρησης υπηρεσιών (για παράδειγμα θα μπορούσε να λειτουργεί ως οθόνη δοσοληψιών για να περιορίσει τον αριθμό των ταυτόχρονων αιτήσεων στον εξυπηρετητή), υπηρεσίες τεχνητής νοημοσύνης (για παράδειγμα υπηρεσίες χαρτογράφησης - εντοπισμού των κατάλληλων εξυπηρετητών για την καλύτερη εξυπηρέτηση της αίτησης του πελάτη, σύγκριση των αποτελεσμάτων που αυτοί επιστρέφουν και επιστροφή της τελικής απάντησης στον πελάτη). Συμπληρωματικά αναφέρουμε ότι υπάρχει και η πολυεπίπεδη αρχιτεκτονική *client / server* η οποία δεν αναλύεται περισσότερο.



Σχήμα 4.2:Αρχιτεκτονική *client / server* 3 επιπέδων

4.1.1.5 Μεσισμικό (Middleware)

Ο όρος μεσισμικό *middleware* αναφέρεται στη γενική κατηγορία του λογισμικού που μεσολαβεί προκειμένου να επικοινωνήσουν 2 διεργασίες ανεξαρτήτως τοποθεσίας. Το στοιχείο κλειδί είναι το λειτουργικό σύστημα δικτύου *(Network Operating System, NOS)*. Το *NOS* παρέχει υπηρεσίες όπως δρομολόγηση, κατανεμημένες υπηρεσίες, αποστολή μηνυμάτων, διανομή αρχείων και εκτύπωση αυτών και τέλος υπηρεσίες διαχείρισης του

δικτύου. Το *NOS* βασίζεται σε πρωτόκολλα επικοινωνιών για να παρέχει τις συγκεκριμένες υπηρεσίες. Τα πρωτόκολλα διαιρούνται σε τρεις ομάδες : τα πρωτόκολλα φυσικού επιπέδου, τα πρωτόκολλα επιπέδου μεταφοράς και τα πρωτόκολλα client / server. Τα πρωτόκολλα φυσικού επιπέδου καθορίζουν τον τύπο των φυσικών συνδέσεων που χρησιμοποιούνται σε ένα δίκτυο (μερικά παραδείγματα είναι τα *Ethernet, Token Ring, Fiber Distributed Data Interface (FDDI),* ομοαξονικού και συνεστραμμένου ζεύγους καλωδίου). Τα πρωτόκολλα μεταφοράς παρέχουν το μηχανισμό της μετακίνησης των πακέτων δεδομένων από τον πελάτη στον εξυπηρετητή (μερικά παραδείγματα πρωτοκόλλων μεταφοράς είναι *Novell's IPX/SPX, Apple's AppleTalk, Transmission Control Protocol / Internet Protocol (TCP/IP), Open Systems Interconnection (OSI) και Government Open Systems Interconnection Profile(GOSIP))*. Από την στιγμή που δημιουργείται η φυσική σύνδεση και επιλεχθεί το πρωτόκολλο μεταφοράς απαιτείται και η επιλογή του πρωτοκόλλου *client / server* πριν ο χρήστης να αποκτήσει πρόσβαση στις υπηρεσίες του δικτύου. Ένα *client / server* πρωτόκολλο υπαγορεύει τον τρόπο σύμφωνα με τον οποίο οι πελάτες θα ζητάνε πληροφορίες μέσω των αιτήσεων τους από τις υπηρεσίες που παρέχουν οι εξυπηρετητές και ορίζουν και τον τρόπο σύμφωνα με τον οποίο οι εξυπηρετητές θα απαντάνε στις αιτήσεις των πελατών (μερικά παραδείγματα πρωτοκόλλων *client / server είναι τα NetBIOS, RPC, Advanced Program-to-Program Communication (APPC), Named Pipes, Sockets, Transport Level Interface (TLI) και Sequenced Packet Exchange (SPX)).*

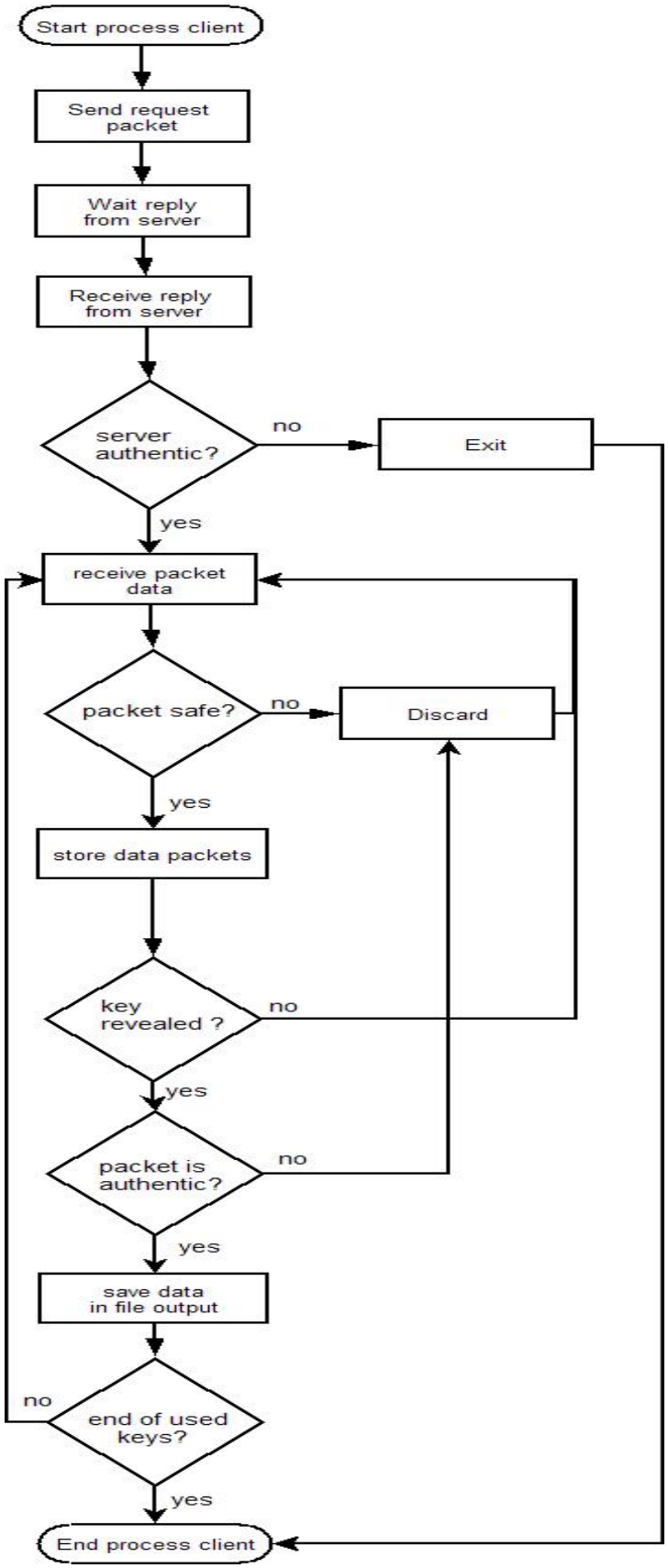
4.2 Περιγραφή Λειτουργιών

Σκοπός αυτή της ενότητας είναι η περιγραφή των λειτουργιών των υποσυστημάτων του *TESLA* όπως αυτό υλοποιήθηκε στη γλώσσα JAVA. Εφόσον το σύστημα μας αποτελείται από δύο υποσυστήματα, του client και του server ακολουθεί η περιγραφή τους.

4.2.1 Υποσύστημα client

Ο *client* στέλνει ένα πακέτο – αίτηση στον *server* (ο οποίος "τρέχει" σε γνωστή πόρτα του συστήματος) με το οποίο ουσιαστικά δηλώνει την επιθυμία του να λάβει τις πληροφορίες που θα του στείλει ο *server.* Το αρχικό αυτό πακέτο είναι διαμορφωμένο σύμφωνα με τις προδιαγραφές του πρωτοκόλλου όπως έχει περιγραφή στην ενότητα 3.2 .Κατόπιν ο *client* περιμένει για την απάντηση του *server.* Όταν λάβει την απάντηση από τον *server,* ο *client* ελέγχει αν το πακέτο που έλαβε προέρχεται πράγματι από τον *server.* Αν αυτό ισχύει τότε μπαίνει στη διαδικασία λήψης των πακέτων δεδομένων σύμφωνα πάντα με τους κανόνες του πρωτοκόλλου. Δηλαδή αρχικά αποφασίζει αν τα πακέτα είναι ασφαλή. Αν είναι τα αποθηκεύει προσωρινά μέχρι να αρχίσουν να αποκαλύπτονται τα κλειδιά. Όταν αποκαλύπτονται τα κλειδιά τότε αυθεντικοποιεί τα πακέτα δεδομένων. Αν τα πακέτα είναι αυθεντικά τότε αποθηκεύει τα δεδομένα σε ένα αρχείο εξόδου. Αν τα πακέτα δεν είναι

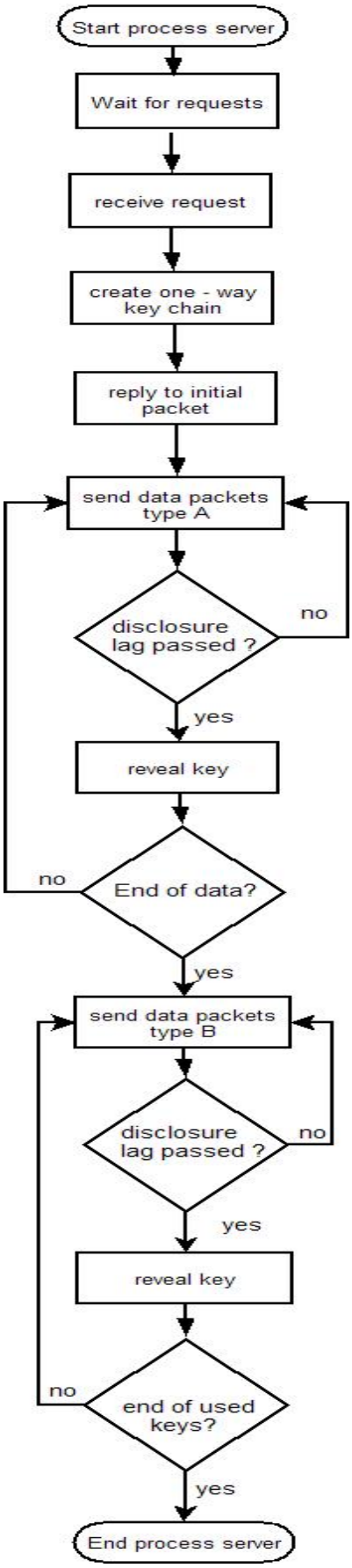
ασφαλή ή δεν είναι αυθεντικά τα απορρίπτει. Ακολουθεί ένα διάγραμμα ροής *(flow chart)* το οποίο περιγράφει την παραπάνω διαδικασία (σχήμα 4.3).



***Σχήμα 4.3*:**Η διεργασία client

4.2.2 Υποσύστημα server

Ο *server* έχει στη διάθεση του τις πληροφορίες που θα στείλει στον *client* και περιμένει έως ότου δεχτεί κάποια αίτηση από αυτόν. Όταν δεχτεί το πακέτο – αίτηση τότε δημιουργεί ένα ικανοποιητικό αριθμό κλειδιών τα οποία θα χρησιμοποιηθούν για την αυθεντικοποίηση των πακέτων δεδομένων μέσω της παραγωγής των κωδικών αυθεντικοποίησης *(MACs)*. Κατόπιν του απαντάει με ένα άλλο πακέτο συγκεκριμένης δομής η οποία ορίζεται από τους κανόνες του πρωτοκόλλου. Στη συνέχεια αρχίζει να στέλνει δεδομένα στον *client* αποκαλύπτοντας τα κλειδιά μετά το πέρας της καθυστέρησης αποκάλυψης. Όταν τελειώσουν τα δεδομένα συνεχίζει να στέλνει πακέτα μέχρι να αποκαλυφθούν όλα τα κλειδιά που χρησιμοποιήθηκαν κρατώντας πάντα την καθυστέρηση αποκάλυψης και να μπορέσει έτσι ο *client* να κάνει την αυθεντικοποίηση. Ακολουθεί ένα διάγραμμα ροής *(flow chart)* το οποίο περιγράφει την παραπάνω διαδικασία (σχήμα 4.4).



***Σχήμα 4.4*:** Η διεργασία server

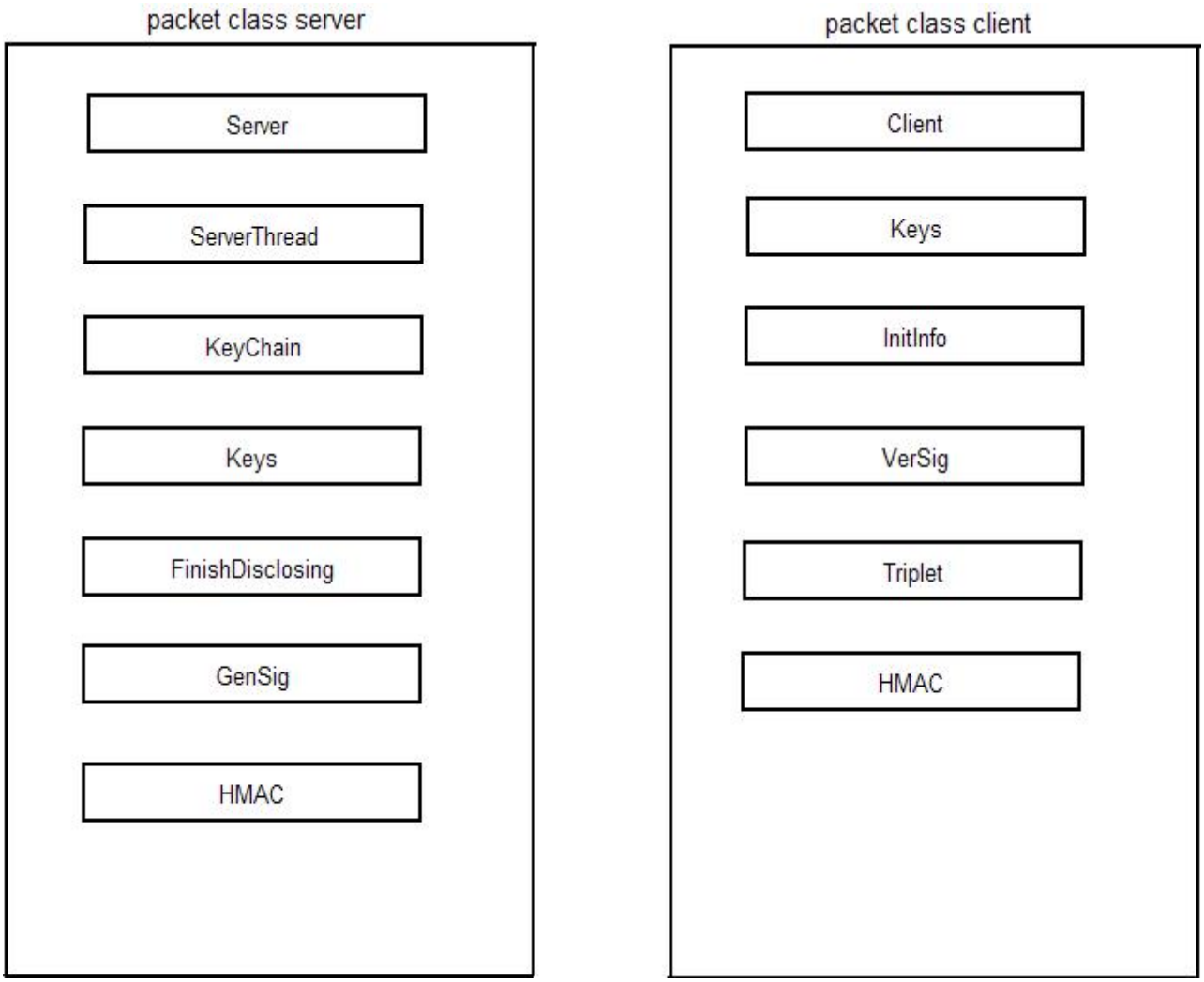
5

Σχεδίαση Συστήματος

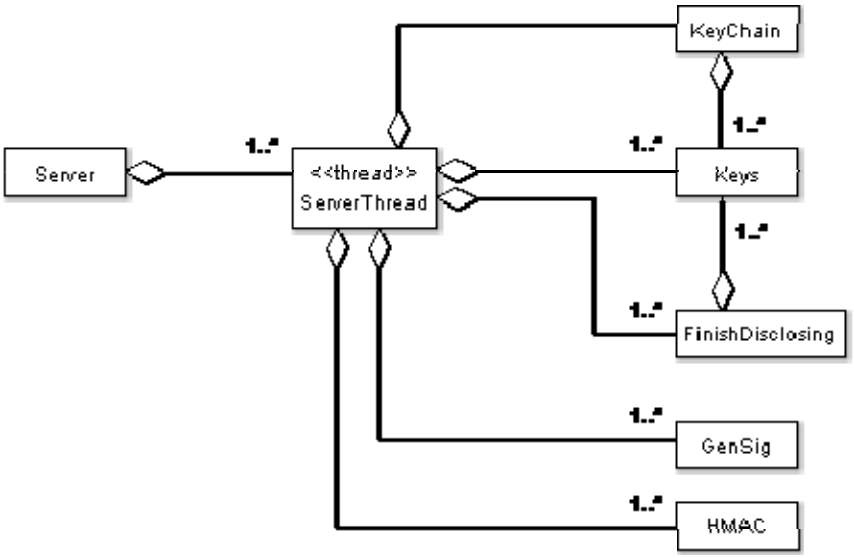
Σε αυτό το κεφάλαιο θα δοθεί η σχεδίαση του συστήματος, παρουσιάζοντας τα επιμέρους κομμάτια από τα οποία έχει κτιστεί ο κώδικάς μας. Τα κομμάτια αυτά είναι στην ουσία οι κλάσεις του συστήματος *TESLA*. Ακολουθούν σχήματα στα οποία αναπαριστάνονται τα πακέτα του συστήματος (πακέτα client / server), το διάγραμμα κλάσεων του κάθε πακέτου και μια σύντομη περιγραφή για κάθε κλάση.

5.1 Πακέτα κλάσεων

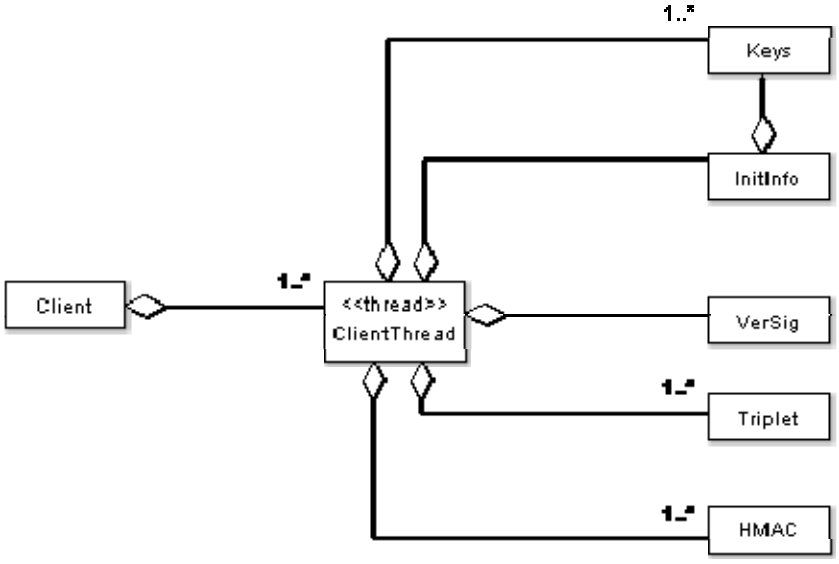
Στο σύστημα υπάρχουν δυο πακέτα κλάσεων, ένα για τον εξυπηρετητή *(server)* και ένα για τον πελάτη *(client)*. Στο παρακάτω σχήμα φαίνονται ποιες κλάσεις περιέχει το κάθε πακέτο (σχήμα 5.1). Ακόμη δίνονται τα σχήματα των διαγραμμάτων κλάσεων των πακέτων.



Σχήμα 5.1:Πακέτα κλάσεων server / client



Σχήμα 5.2:Διάγραμμα κλάσεων πακέτου server



Σχήμα 5.2:Διάγραμμα κλάσεων πακέτου client

5.2 Περιγραφή Κλάσεων

Ακολουθεί συνοπτική περιγραφή των λειτουργιών / μεθόδων των κλάσεων του κάθε πακέτου.

5.2.1 Πακέτο server

5.2.1.1 Public class Server

Η κλάση αυτή αποτελεί την βασική κλάση του *server*, η οποία με τη σειρά της δημιουργεί τα server Thread που θα εξυπηρετήσουν τους *client* (ουσιαστικά τα αντίστοιχα client Thread).

Μέθοδοι:

- public Server()

  Constructor της κλάσης
- public static void main(String[] args)

  Η main είναι η βασική συνάρτηση της κλάσεως *server* η οποία περιέχει όλες τις απαραίτητες πληροφορίες για τη λειτουργία του, όπως σε ποια θύρα του συστήματος ‘ακούει’, το μήκος της αλυσίδας των κλειδιών, το μέγεθος των χρονικών διαστημάτων, τον αριθμό των Server Thread, την καθυστέρηση της αποκάλυψης των κλειδιών, τον αριθμό των Server Thread που δημιουργούνται (μέγεθος της thread Pool) και τον αριθμό των Server Thread που εκτελούνται.

5.2.1.2 Public class ServerThread

Υλοποιεί την διαπροσωπεία (interface) runnable προκειμένου τα αντικείμενα αυτής της κλάσης να εκτελούνται από νήματα (threads).

Πεδία :

- static double reset = 1.135296E12;
  Η σταθερά reset εκφράζει πόσα milliseconds πέρασαν από το 1970 μέχρι το 2006.
- static int packetSize=128;
  Το μέγεθος των πακέτων που στέλνει ο *server* σε bytes.
- static String ts;
  Η τοπική ώρα του server.
- String NameThread;
  Το όνομα του Thread.
- int keyChainLength;
  Το μήκος της αλυσίδας των κλειδιών.
- int indOfInterv;
  Ο δείκτης αρίθμησης των χρονικών διαστημάτων.
- int disclosureLag;
  Καθυστέρηση αποκάλυψης των κλειδιών.
- int Tint;
  Χρονική διάρκεια των διαστημάτων.
- int count2=0;
  Ο count2 παριστάνει την αρίθμηση του κλειδιού που θα αποκαλυφθεί στον client. Αυξάνεται όταν περάσει η καθυστέρηση αποκάλυψης των κλειδιών. Χρησιμοποιείται από την time corresponding key.
- int count1=0;
  Ο count1 παριστάνει πόσα διαστήματα περνάνε. Αν περάσουν τα διαστήματα που ορίζει η disclosure lag αποκαλύπτεται το αντίστοιχο κλειδί στον client .
- double Ti;
  Η χρονική στιγμή έναρξης της λειτουργίας του *server.*
- Keys activeKey1;
  Το activeKey1 είναι που αποκαλύπτεται στον client. Παίρνει τιμές ανάλογα σε ποιο χρονικό διάστημα είναι ο *server*. Για παράδειγμα για 0<t<disclosure lag θα είναι K0 ενώ για disclosure lag< t < 2 * (disclosure lag) θα είναι K1.
- Keys activeKey2;

Το activeKey2 χρησιμοποιείται για τον υπολογισμό της *MAC* του μηνύματος.

- String curDir;

  Περιέχει το working directory του project.

- String fileSep;

  Περιέχει τον χαρακτήρα file separator του συστήματος.

Μέθοδοι:

- public ServerThread(String str,int socketport,int key_chain_length,int ind_of_interv,int disclosure_lag,double T_i,int T_int)

  Ο constructor της κλάσης αυτής. Θέτει τις τιμές των ορισμάτων του στα πεδία της κλάσης.

- private String getTime();

  Επιστρέφει σ' ένα String την ώρα του συστήματος .Πιο συγκεκριμένα μετράει πόσα milliseconds πέρασαν από την αρχή του 2006 μέχρι τη στιγμή μέτρησης.

- public void run();

  Είναι η καρδιά του Server Thread . Είναι η μέθοδος που καλείται πρώτη και εδώ υλοποιείται ουσιαστικά ο server

- private String getName();

  Επιστρέφει το όνομα του Server Thread

- private String formatKey(String key);

  Διαμορφώνει τη δομή του κλειδιού που δέχεται ως όρισμα. Το κλειδί έχει μήκος μέχρι και 9 ψηφία. Ανήκει δηλαδή στο εύρος τιμών 000000000-999999999.Επιστρέφει το διαμορφωμένο κλειδί ως string.

- private String formatDisclosureLag(int disclosureLag);

  Διαμορφώνει τη δομή της καθυστέρησης αποκάλυψης την οποία δέχεται ως όρισμα. Η καθυστέρηση αποκάλυψης έχει μήκος μέχρι και 2 ψηφία. Ανήκει δηλαδή στο εύρος τιμών 00-99.Επιστρέφει τη διαμορφωμένη καθυστέρηση ως string.

- private String formatKeyChainLength(int keyChainLength);

  Διαμορφώνει τη δομή το μήκος της αλυσίδας κλειδιών το οποίο δέχεται ως όρισμα. Το μήκος της αλυσίδας κλειδιών είναι μέχρι και 4 ψηφία. Ανήκει δηλαδή στο εύρος τιμών 0000-9999. Επιστρέφει το διαμορφωμένο μήκος ως string.

- private String formatTint(int Tint);

  Διαμορφώνει τη δομή της διάρκειας του χρονικού διαστήματος το οποίο δέχεται ως όρισμα. Το χρονικό διάστημα μπορεί να είναι μέχρι και 4 ψηφία. Ανήκει δηλαδή στο εύρος τιμών 0000-9999. Επιστρέφει το διαμορφωμένη διάρκεια ως string.

- private String formatInterv(int ind_of_interv) ;

Διαμορφώνει την αρίθμηση του χρονικού διαστήματος το οποίο δέχεται ως όρισμα έτσι ώστε η μέγιστη τιμή του να είναι 65535 (4 bytes FFFF) και να είναι και στη μορφή 5 ψηφίων (π.χ. 00012 ή 000512) . Η αρίθμηση μπορεί να είναι μέχρι και 5 ψηφία. Ανήκει δηλαδή στο εύρος τιμών 00000-65535. Επιστρέφεται η αρίθμηση του χρονικού διαστήματος ως string.

- public void printKeys(ArrayList keyList);
  Τυπώνει τα κλειδιά που περιέχει η keyList
- public void timeCorrespondingKey(ArrayList keyList,ArrayList keyMac,KeyChain keyChain);
  Θέτει τιμές στα activeKey1 και activeKey2 αναλόγως σε ποιο χρονικό διάστημα βρίσκεται ο *server*. Τα activeKey1 και activeKey2 είναι τα κλειδιά που αποκαλύπτονται στον client και τα κλειδιά που χρησιμοποιούνται για την παραγωγή των MACs.Λαμβάνει τα κλειδιά από τις KeyMac και keyChain. Ουσιαστικά οι KeyMac και keyChain αποτελούν τις F , F' που αναφέρονται στο θεωρητικό υπόβαθρο. Ακόμη αυξάνει και την αρίθμηση των χρονικών διαστημάτων όταν περάσει χρόνος ίσος με τη χρονική διάρκεια Tint.
- private String constructPacket(String Mj,int i,Keys Ki,String Mac);
  Η constructPacket κατασκευάζει τα πακέτα αποστολής σύμφωνα με τα ορίσματα που της δίνονται. Το πρώτο όρισμα είναι το μήνυμα που θα μεταδοθεί (data), το δεύτερο το διάστημα στο οποίο ανήκει το πακέτο, το τρίτο το κλειδί που αποκαλύπτεται και το τέταρτο ο κωδικός αυθεντικότητας του.

5.2.1.3 public class Keys

Η κλάση υλοποιεί την interface SecretKey.

Πεδία:

- public int sel;
  Είναι η τιμή του κλειδιού. Περιέχεται από κάθε αντικείμενο της κλάσης. Όταν αναφερόμαστε σε ένα κλειδί ουσιαστικά αναφερόμαστε στο πεδίο αυτό.

Μέθοδοι:

- public Keys(int sel);
  Ο constructor της κλάσης. Θέτει το atrribute sel στην τιμή του ορίσματος του.
- public byte[] getEncoded() ;
  Επιστρέφει την κωδικοποίηση του κλειδιού σε bytes. Ανήκει στην interface SecretKey.
- public String getFormat();

Επιστρέφει το όνομα της κωδικοποίησης αυτού του κλειδιού. Ανήκει στην interface SecretKey.

- public String getAlgorithm();
  Επιστρέφει το όνομα του αλγορίθμου για αυτό το κλειδί. Ανήκει στην interface SecretKey.

5.2.1.4 public class KeyChain

Πεδία:

- protected ArrayList keyList;
  Η ArrayList στην οποία σώζονται τα κλειδιά. Είναι τα Ki.
- protected ArrayList keyMac;
  Τα κλειδιά που σώζονται στην keyMac χρησιμοποιούνται για τον υπολογισμό των MACs. Είναι τα Ki΄.
- protected Random rand;
  Αντικείμενο της κλάσης Rand. Χρησιμοποιείται για την παραγωγή ψευδοτυχαίων αριθμών.
- static String NameThread;
  Το όνομα του Server Thread.

Μέθοδοι:

- public KeyChain(String NameThread);
  Ο constructor της κλάσης. Θέτει το atrribute NameThread στην τιμή του ορίσματος του.
- public ArrayList keys();
  Υπολογίζει τις ακολουθίες των κλειδιών και τις αποθηκεύει στις ArrayList.

5.2.1.5 public class HMAC

Πεδία:

- String datamessage;
  Το μήνυμα για το οποίο θα υπολογιστεί ο MAC.

Μέθοδοι:

- public HMAC(String datamessage);
  Ο constructor της κλάσης. Θέτει το atrribute datamessage στην τιμή του ορίσματος του.
- public String generateMac(Keys key);
  Κατασκευάζει τον MAC από το κλειδί που δίνεται ως όρισμα. Ο MAC επιστρέφεται ως string.

5.2.1.6 public class GenSig

Η κλάση GenSig δημιουργεί ένα ζευγάρι κλειδιών (ιδιωτικό και δημόσιο) και τα σώζει σε ένα αρχείο. Ακόμη σώζει σε ένα άλλο αρχείο την υπογραφή των δεδομένων που βρίσκονται στον πίνακα που δίνεται σαν όρισμα της, δηλαδή τον buffer.

Πεδία:

- String NameThread;
  Το όνομα του Server Thread.
- int port;
  Η θύρα του συστήματος στην οποία τρέχει ο *server*.
- static byte[] buffer;
  Περιέχει τα δεδομένα που θα υπογραφούν.
- boolean signed=false;
  Boolean μεταβλητή. Παίρνει τιμη true αν υπογραφούν τα δεδομένα το buffer επιτυχώς.

Μέθοδοι:

- public GenSig(byte[] buffer,int port,String NameThread)
  Ο constructor της κλάσης. Θέτει τα atrributes στις τιμές των ορισμάτων του. Ακόμη καλεί την μέθοδο genSig.
- public boolean genSig(String NameThread)
  Η genSig υπογράφει τα δεδομένα του buffer και εκτελεί τις κύριες λειτουργίες της κλάσης.

5.2.1.7 public class FinishDisclosing

Η FinishDisclosing αποκαλύπτει τα κλειδιά που απομένουν μετά τη χρονική στιγμή που τελειώνουν τα δεδομένα προς αποστολή. Σε αυτήν την περίπτωση τα πακέτα διατηρούν την δομή που είχαν, αλλά στις θέσεις των δεδομένων και του MAC γράφεται ένα string μηδενικού ενδιαφέροντος. Αυτό που ενδιαφέρει τον client είναι το κλειδί και μόνο. Τα κλειδιά αποκαλύπτονται πάντα με διατήρηση του αρχικού χρονοδιαγράμματος και της καθυστέρησης αποκάλυψης των κλειδιών.

Πεδία:

- static double reset = 1.135296E12;
  Η σταθερά reset εκφράζει πόσα milliseconds πέρασαν από το 1970 μέχρι το 2006.
- String MAC = "000000000000000000000000";
  Ο MAC εχει την σταθερή ειδική αυτή τιμή αφού δεν δημιουργούνται πλέον νέες τιμές. Απλά διατηρείται η δομή των 24 ψηφίων.

- int saveint;

  Ο συνολικός αριθμός των κλειδιών που χρησιμοποιήθηκαν.
- int count2;

  Μετρητής που αριθμεί τα κλειδιά που αποκαλύπτονται.
- int keyChainLength;

  Το μήκος της αλυσίδας των κλειδιών.
- int Tint;

  Χρονική διάρκεια των διαστημάτων.
- int indOfInterv;

  Ο δείκτης αρίθμησης των χρονικών διαστημάτων.
- int count1;

  Μετρητής που αριθμεί διαστήματα. Αν περάσουν count1 = disclosure lag διαστήματα αποκαλύπτεται το επόμενο κλειδί, αυξάνεται το count2 και το count1 γίνεται reset στην τιμή 0. Ως απόρροια αυτής της λογικής το count παίρνει τιμές στο δίαστημα [0,disclosure lag].
- int disclosureLag;

  Καθυστέρηση αποκάλυψης των κλειδιών.
- int port;

  Η θύρα που τρέχει ο server.
- ArrayList keyList;

  Η ArrayList στην οποία σώζονται τα κλειδιά. Είναι τα Ki.
- Double curTs;

  Ο τρέχων χρόνος του server.
- Keys activeKey1;

  Το activeKey1 είναι που αποκαλύπτεται στον client. Παίρνει τιμές ανάλογα σε ποιο χρονικό διάστημα είναι ο *server*.
- InetAddress address;

  Παριστάνει τη διεύθυνση του client.Είναι η διεύθυνση στην οποία στέλνονται τα πακέτα
- DatagramSocket socket;

  Το μεσισμικό με το οποίο θα σταλούν τα πακέτα. Είναι το κανάλι επικοινωνίας του server με τον client.

Μέθοδοι:

- public FinishDisclosing(int saveint,DatagramSocket socket,ArrayList keyList,Double curTs,int key_chain_length,int Tint,int indOfInterv,int count1,int disclosureLag,int count2,Keys activeKey1,InetAddress address,int port,String NameThread)
  Ο constructor της κλάσης. Θέτει τα atrributes της κλάσης στις τιμές των ορισμάτων του.
- public int discloseRestOfKeys(ArrayList keyList)
  Η μέθοδος αποκάλυψης των κλειδιών.
- private String getTime()
  Επιστρέφει τον τρέχον χρόνο του τη στιγμή της κλήσης της.
- private String constructPacket(String Mj,int i,Keys Ki,String Mac)
  Κατασκευάζει το πακέτο που θα σταλεί από τα ορίσματα της. Το πακέτο έχει τη δομή string.
- private String formatKey(String key)
  Διαμορφώνει τη δομή του κλειδιού που δέχεται ως όρισμα. Το κλειδί έχει μήκος μέχρι και 9 ψηφία. Ανήκει δηλαδή στο εύρος τιμών 000000000-999999999.Επιστρέφει το διαμορφωμένο κλειδί ως string.
- private String formatInterv(int ind_of_interv)
  Διαμορφώνει την αρίθμηση του χρονικού διαστήματος το οποίο δέχεται ως όρισμα έτσι ώστε η μέγιστη τιμή του να είναι 65535 (4 bytes FFFF) και να είναι και στη μορφή 5 ψηφίων (π.χ. 00012 ή 000512) . Η αρίθμηση μπορεί να είναι μέχρι και 5 ψηφία. Ανήκει δηλαδή στο εύρος τιμών 00000-65535. Επιστρέφεται η αρίθμηση του χρονικού διαστήματος ως string.
- public void sendKeys(int packetSize)
  Στέλνει τα κλειδιά μέσω των πακέτων αποκάλυψης.

5.2.2 Πακέτο client

5.2.2.1 Public class Client

Η κλάση αυτή αποτελεί την βασική κλάση του *client*, η οποία με τη σειρά της δημιουργεί τα client Thread που θα επικοινωνήσουν με τους server (ουσιαστικά τα αντίστοιχα Server Thread).

Μέθοδοι:

- public static void main(String[] args)

Η main είναι η βασική συνάρτηση της κλάσεως *client* η οποία περιέχει απαραίτητες πληροφορίες για τη λειτουργία του, όπως σε ποια θύρα του συστήματος 'ακούει' ο *server*, την θύρα στην οποία 'τρέχει' ο client, τον αριθμό των Client Thread που δημιουργούνται (μέγεθος της thread Pool) ,το όνομα του κάθε Client Thread και τον αριθμό των Client Thread που εκτελούνται.

5.2.2.2 Public class ClientThread

Υλοποιεί την διαπροσωπεία (interface) runnable προκειμένου τα αντικείμενα αυτής της κλάσης να εκτελούνται από νήματα (threads).

Πεδία:

- String argument;
  Το μηχάνημα στο οποίο τρέχει ο server. Εδώ είναι ο localhost.
- static int length=32;
  Το μέγεθος του πίνακα που περιέχουν τα δεδομένα.
- int socketport;
  Η πόρτα που 'τρέχει' ο client
- int disclosureLag;
  Καθυστέρηση αποκάλυψης
- ArrayList KeyList;
  ArrayList στην οποία σώζονται τα κλειδιά που αποκαλύπτονται. Είναι τα Ki.
- ArrayList KeyMac;
  ArrayList στην οποία σώζονται τα κλειδιά που θα χρησιμοποιηθούν για την επαλήθευση των MACs. Είναι τα Ki΄.
- ArrayList bufferPackets;
  ArrayList που σώζονται αντικείμενα – πακέτα προκειμένου να αυθεντικοποιηθούν μετά την αποκάλυψη των κλειδιών.
- Keys activeKey;
  Παριστάνει το κλειδί που αποκαλύπτεται
- Random randR;
  Χρησιμοποιείται για την παραγωγή των μονόδρομων αλυσίδων
- double reset = 1.135296E12;
  Η σταθερά reset εκφράζει πόσα milliseconds πέρασαν από το 1970 μέχρι το 2006.
- String NameThread;
  Το όνομα του Client Thread

Μέθοδοι:

- public ClientThread(String str,String arg,int socketport)

Ο constructor της κλάσης. Θέτει τα atrributes της κλάσης στις τιμές των ορισμάτων του.

- public void run()
  Είναι η μέθοδος που εκτελεί όλες τις απαραίτητες λειτουργίες του client.
- private int count_length(byte[] data_from_server)
  Μετρά πόσα μηδενικά υπάρχουν στις πρώτες length θέσεις του ορίσματος της. Επιστρέφει το μήκος της απάντησης του server.
- private String getTime()
  Επιστρέφει σ' ένα String την ώρα του συστήματος .Πιο συγκεκριμένα μετράει πόσα milliseconds πέρασαν από την αρχή του 2006 μέχρι τη στιγμή μέτρησης.
- private String getName()
  Επιστρέφει το όνομα του Client Thread
- public InitInfo getInitInfo(String received)
  Επιστρέφει ένα αντικείμενο τύπου InitInfo το οποίο περιέχει όλες τις πληροφορίες που χρειάζονται για το αρχικό πρωτόκολλο συγχρονισμού.
- private String getMAC(byte[] dataBuffer,int packetSize)
  Επιστρέφει σε String τον MAC που περιέχεται στον πίνακα dataBuffer.
- public boolean reConstructChain(Keys activeKey,ArrayList KeyList)
  Ανακατασκευάζει την αλυσίδα των κλειδιών
- public void printKeys(ArrayList keyList)
  Τυπώνει τα κλειδιά που βρίσκονται στην keyList.
- private Keys getKey(byte[] dataBuffer,int packetSize)
  Επιστρέφει σε ένα αντικείμενο Keys το κλειδί που περιέχεται στον πίνακα dataBuffer.
- private int getInterval(byte[] dataBuffer,int packetSize)
  Επιστρέφει σε ένα ακέραιο την αρίθμηση του διαστήματος που εστάλη το πακέτο τα δεδομένα του οποίου περιέχονται στον πίνακα dataBuffer.
- private String getMessage(byte[] dataBuffer,int packetSize)
  Επιστρέφει σε ένα String το μήνυμα (δηλαδή την καθαρή πληροφορία) που περιέχει το πακέτο το οποίο περιέχεται στον πίνακα d.ataBuffer.
- private boolean securityCondition(double Tj,double Dt,double T0,double Td,int i1,double RTT)
  Είναι η συνθήκη ασφάλειας που πρέπει να ικανοποιούν τα πακέτα για να κριθούν ως ασφαλή και να αποθηκευθούν για αργότερη αυθεντικοποίηση
- public boolean verifyCommitment(Keys keyRevel,ArrayList keyList)

Επιστρέφει true αν το κλειδί που αποκαλύπτεται επαληθεύεται από την μονόδρομη αλυσίδα.

- public int getIntervalKey(ArrayList keyList)
  Επιστρέφει την αρίθμηση του αποκαλυφθέντος κλειδιού στην αλυσίδα.

5.2.2.3 Public class HMAC

Σκοπός της κλάσης αυτής είναι η παραγωγή των κωδικών αυθεντικότητας που χρειάζονται για τη εύρυθμη λειτουργία του πρωτοκόλλου.

Πεδία:

- String datamessage;
  Το μήνυμα για το οποίο θα υπολογιστεί ο MAC.

Μέθοδοι:

- public HMAC(String datamessage);
  Ο constructor της κλάσης. Θέτει το atrribute datamessage στην τιμή του ορίσματος του.
- public String generateMac(Keys key);
  Κατασκευάζει τον MAC από το κλειδί που δίνεται ως όρισμα. Ο MAC επιστρέφεται ως string.

5.2.2.4 Public class Keys

Η κλάση υλοποιεί την interface SecretKey.

Πεδία:

- public int sel;
  Είναι η τιμή του κλειδιού. Περιέχεται από κάθε αντικείμενο της κλάσης. Όταν αναφερόμαστε σε ένα κλειδί ουσιαστικά αναφερόμαστε στο πεδίο αυτό.

Μέθοδοι:

- public Keys(int sel);
  Ο constructor της κλάσης. Θέτει το atrribute sel στην τιμή του ορίσματος του.
- public byte[] getEncoded() ;
  Επιστρέφει την κωδικοποίηση του κλειδιού σε bytes. Ανήκει στην interface SecretKey.
- public String getFormat();
  Επιστρέφει το όνομα της κωδικοποίησης αυτού του κλειδιού. Ανήκει στην interface SecretKey.
- public String getAlgorithm();

Επιστρέφει το όνομα του αλγορίθμου για αυτό το κλειδί. Ανήκει στην interface SecretKey.

5.2.2.5 Public class InitInfo

Ορίζει αντικείμενα που περιέχουν όλες τις απαραίτητες πληροφορίες του αρχικού πρωτόκολλου συγχρονισμού.

Πεδία:

- String ts;

  Η ώρα του server.
- Keys key;

  Η αρχική δέσμευση (commitment) του server.
- int key_chain_length;

  Το μήκος της αλυσίδας των κλειδιών.
- String ind_of_interv;

  Η αρίθμηση των χρονικών διαστημάτων.
- int diclosure_lag;

  Η καθυστέρηση αποκάλυψης.
- double T_i;

  Η χρονική στιγμή έναρξης της λειτουργίας του *server*.
- double T_int;

  Η χρονική διάρκεια του κάθε διαστήματος
- String nonce;

  Η nonce που στέλνει αρχικά ο client για τις ανάγκες του αρχικού πρωτοκόλλου συγχρονισμού.

Μέθοδοι:

- public InitInfo(String received,String NameThread)

  Ο constructor της κλάσης. Φιλτράρει τις αρχικές πληροφορίες από το όρισμα received και τις αποθηκεύει στης μεταβλητές της κλάσης.

5.2.2.6 Public class Triplet

Η κλάση αυτή αποτελεί τη δομή σύμφωνα με την οποία αποθηκεύονται τα πακέτα μέχρι να επαληθευτούν.

Πεδία:

- int interval;

  Το διάστημα στο οποίο εστάλη το πακέτο.
- String Mj;

Το μήνυμα που περιέχει το πακέτο.

- String MAC;
  Ο κωδικός αυθεντικοποίησης του πακέτου.

Μέθοδοι:

- public Triplet(int interval,String Mj,String MAC)
  Ο πρώτος constructor της κλάσης. Θέτει όλα τα attributes της κλάσης στις τιμές των ορισμάτων του.
- public Triplet()
  Ο δεύτερος constructor της κλάσης. Θέτει όλα τα attributes της κλάσης σε μηδενικές τιμές.

5.2.2.7 Public class VerSig

Χρησιμοποιείται από τον *Client* για την επαλήθευση της ψηφιακής υπογραφή του *Server*.

Πεδία:

- boolean verifies
- String NameThread;
  Το όνομα του Client Thread
- byte[] buf;
  Περιέχει τα δεδομένα προς επαλήθευση.
- int port;
  Η θύρα στην οποία 'τρέχει' ο client

Μέθοδοι:

- public VerSig(byte[] buf, int port,String NameThread)
  Ο constructor της κλάσης. Θέτει το τα attributes της κλάσης στις τιμές των ορισμάτων του.
- public void verSig(String NameThread)
  Επαληθεύει την υπογραφή του server θέτοντας τη boolean μεταβλητή verifies στην τιμή true ή false.
- public static void printarray(byte[] data)
  Τυπώνει τα δεδομένα του πίνακα data.

5.3 Κωδικοποίηση αρχείων

Το σύστημα *TESLA* όπως έχει υλοποιηθεί υποστηρίζει την ύπαρξη πολλών *servers* και *clients* συγχρόνως στο δίκτυο. Προκειμένου να επαληθευτεί η ψηφιακή υπογραφή του κάθε *server* για το αρχικό πακέτο, αρκεί ο *client* να εισάγει την ψηφιακή υπογραφή του *server* και να την επαληθεύει με το δημόσιο κλειδί. Ο μηχανισμός σύμφωνα με τον οποίο ο *client* εισάγει αυτές τις πληροφορίες θα μπορούσε να είναι οποιοσδήποτε. Στη συγκεκριμένη υλοποίηση ο *server* σώζει την υπογραφή του αρχικού πακέτου και το δημόσιο κλειδί του σε δυο αρχεία. Η ψηφιακή υπογραφή σε ένα αρχείο ονόματος sig + (την θύρα στην οποία τρέχει ο *server*) και το δημόσιο κλειδί σε ένα αρχείο ονόματος suepk + (την θύρα στην οποία τρέχει ο *server*). Τα αρχεία τοποθετούνται σε γειτονικούς φακέλους από τον φάκελο του *server*. Επομένως ο *client* αφού γνωρίζει σε ποιο *server* έστειλε το αρχικό πακέτο, γνωρίζει τη θύρα του και δεν έχει παρά να ανοίξει το αντίστοιχο αρχείο και να εκτελέσει τη ζητούμενη επαλήθευση. Στο κεφάλαιο 7 και στο σχήμα 7.8 αναπαριστάνεται η παραπάνω διαδικασία.

6

Υλοποίηση

Στο παρόν κεφάλαιο θα αναφέρουμε με περισσότερη λεπτομέρεια θέματα της υλοποίησης.

6.1 Πλατφόρμες και προγραμματιστικά εργαλεία

Η εργασία υλοποιήθηκε στην γλώσσα προγραμματισμού JAVA με jdk1.5.0_10, χρησιμοποιώντας τα περιβάλλοντα Netbeans 5.0 και Netbeans 5.5. Για την υλοποίηση χρησιμοποιήθηκαν τα εξής συστήματα : PC με AMD Athlon XP Processor στα 3000+ και μνήμη ram 512Mb με λειτουργικό Microsoft Windows XP Home Edition, PC με Sempron Processor 3000+ στα 1,8 GHz και μνήμη ram 1Gb με λειτουργικό Microsoft Windows XP Professional, PC με AMD Athlon XP+ Processor στα 1,7 GHz και μνήμη ram 256 Mb με λειτουργικό Microsoft Windows XP Professional.

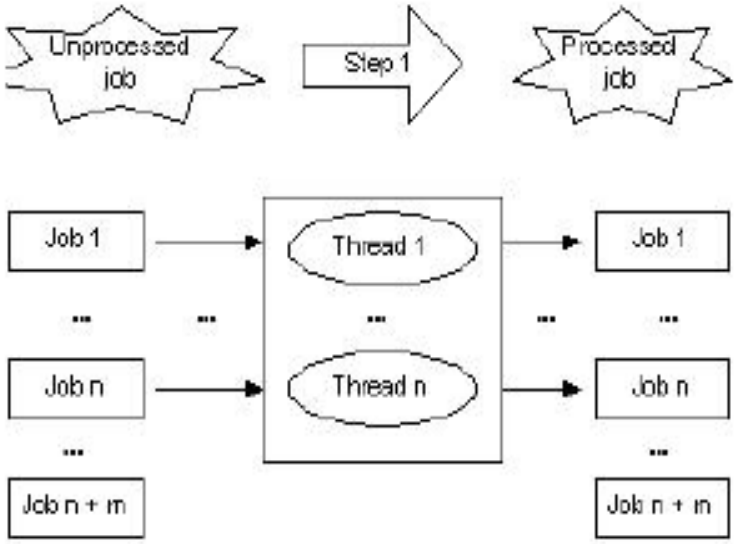
6.2 Λεπτομέρειες υλοποίησης

Στην παρούσα ενότητα θα αναφέρουμε κάποια σημεία της διπλωματικής όπου θεωρούμε ότι υπάρχει αλγοριθμικό ενδιαφέρον. Η περιγραφή θα γίνεται δίνοντας διαγράμματα ροής, τμήματα ψευδοκώδικα ή κώδικα αν αυτό είναι απαραίτητο.

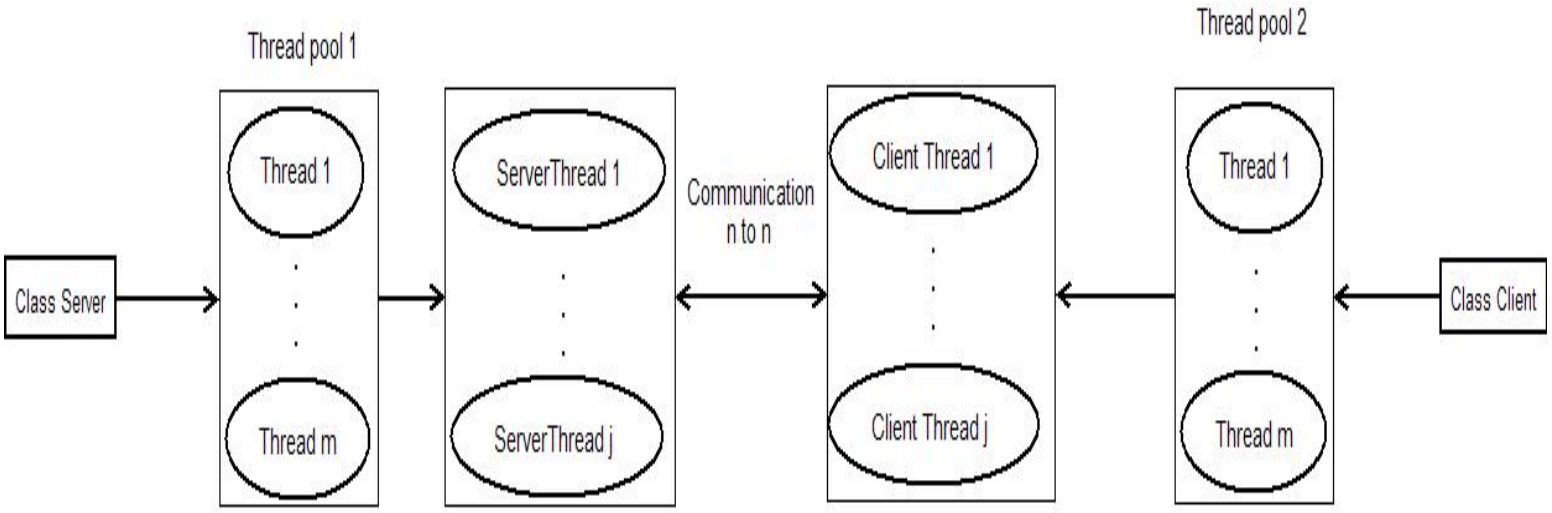
6.2.1 Η τεχνική των thread pools

Τα υποσυστήματα τόσο του client όσο και του server έχουν υλοποιηθεί σύμφωνα με την τεχνική των thread pools. Δηλαδή για να επιτευχθεί η παράλληλη και σύγχρονη εξυπηρέτηση πολλών πελατών στο σύστημα, δημιουργούνται πολλά ServerThreads και πολλά ClientThread. Πρακτικά αυτό σημαίνει ότι υπάρχουν πολλοί servers και clients στο σύστημα. Είναι δηλαδή μια σχέση πολλά προς πολλά. Πιο αναλυτικά και για το υποσύστημα του server μπορούμε να αναφέρουμε ότι υπάρχει η κλάση server η οποία δημιουργεί μια thread pool με τα έτοιμα προς εκτέλεση threads. Η thread pool είναι μια συλλογή από threads, τα οποία έχουν δημιουργηθεί μαζικά. Ο αριθμός των threads που δημιουργούνται εκφράζεται από το μέγεθος της thread pool (έστω N). Οι εφαρμογές (έστω M), που καλούνται να εκτελέσουν τα threads οργανώνονται σε μια ουρά *(queue)*. Συνήθως είναι N<M. Όταν ένα thread τελειώσει την εφαρμογή του, τότε ζητάει από την ουρά την επόμενη προς εφαρμογή προς εκτέλεση. Η

διαδικασία αυτή συνεχίζεται μέχρις ότου αδειάσει η ουρά των εφαρμογών που περιμένουν. Το μέγεθος της thread pool δίνεται από τον χρήστη του συστήματος σε command line ως η δεύτερη παράμετρος. Η πρώτη παράμετρος που δίνει ο χρήστης είναι ο αριθμός των ServerThreads που εκτελούνται. Αυτά είναι ένα υποσύνολο των threads από αυτά που ανήκουν στην thread pool. Για λόγους πληρότητας αναφέρουμε και την εντολή εκτέλεσης του server. Από command line γράφουμε java –jar Server.jar j m όπου 0<j<m. Για παράδειγμα java –jar Server.jar 3 10.Η λογική υλοποίησης του client είναι παρόμοια. Υπάρχει η βασική κλάση client η οποία δημιουργεί μια thread pool του μεγέθους που έχει ορίσει ο χρήστης από την command line. Κατόπιν δημιουργούνται ClientThread πλήθους ίσου με τη δεύτερη παράμετρο που έχει δώσει ο χρήστης. Η μόνη διαφορά στην εκτέλεση του υποσυστήματος client είναι η ύπαρξη μιας παραπάνω παραμέτρου στην σύνταξη της εντολής που δίνει ο χρήστης η οποία παριστάνει το μηχάνημα όπου τρέχει ο server. Για λόγους πληρότητας αναφέρουμε τη σύνταξη της. Από command line γράφουμε java –jar Client.jar "address of server" j m όπου 0<j<m. Για παράδειγμα java –jar Client.jar localhost 3 10. Πρέπει να αναφέρουμε ότι η επικοινωνία μεταξύ των πελατών και εξυπηρετητών του συστήματος, ουσιαστικά γίνεται μεταξύ των ServerThread και των ClientThread.Στο σχήμα που δίνεται παρακάτω απεικονίζονται τα προαναφερθέντα (σχήμα 6.1και 6.2).



***Σχήμα 6.1*:** Η τεχνική thread pool



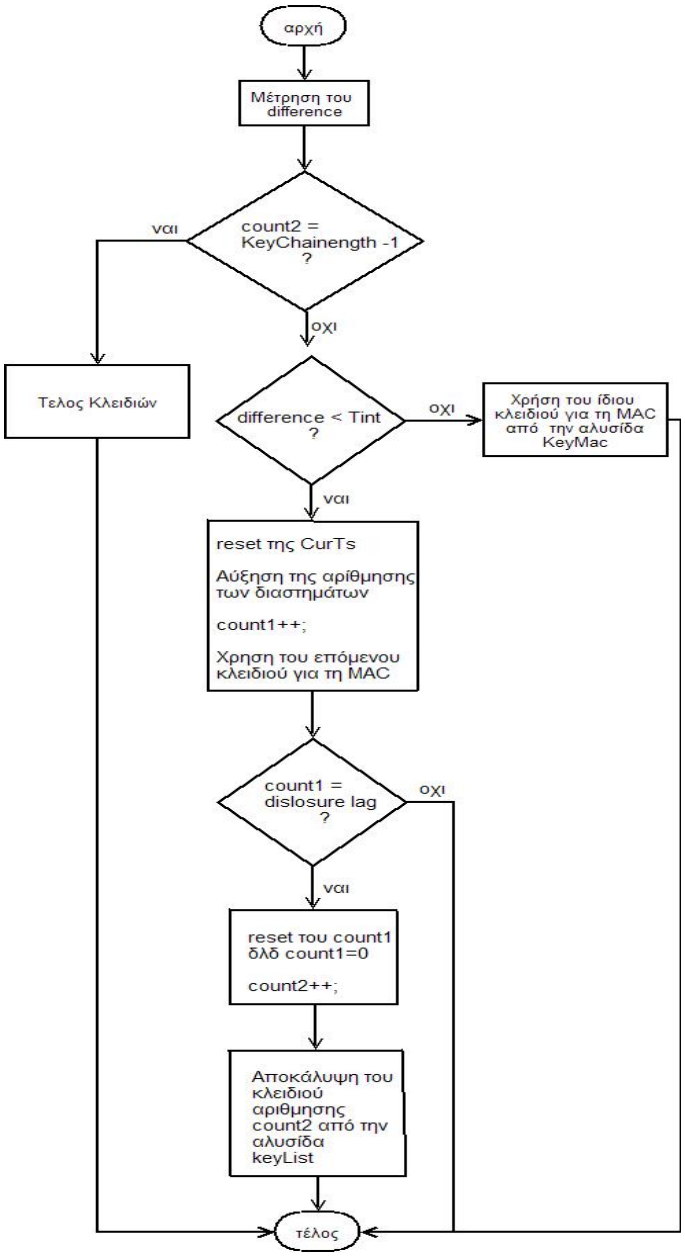
***Σχήμα 6.2*:**Εφαρμογή της τεχνικής thread pool.

Ο κώδικας που υλοποιεί το παραπάνω μοντέλο είναι:

```
ExecutorService tpes = Executors.newFixedThreadPool(threadPoolSize);
 ServerThread[] servers = new ServerThread[numOfServers];
  for(int i=0;i<numOfServers;i++)
   {
       servers[i] = new ServerThread("ServerThread"+i, socketport, keyChainLength,
        indOfInterv , disclosureLag, Ti, Tint);
        tpes.execute(servers[i]);
        socketport++;
    }
```

6.2.2 Μέθοδος timeCorrespondingKey

Σκοπός της μεθόδου είναι η αντιστοίχηση των κλειδιών σύμφωνα με τα χρονικά διαστήματα όπως ορίζει το πρωτόκολλο. Το διάγραμμα ροής της μεθόδου δίνεται παρακάτω (σχήμα 6.3). Πρέπει να αναφέρουμε ότι τόσο στην timeCorrespondingKey όσο και στην DiscloseRestOfKeys η αποκάλυψη των κλειδιών γίνεται λαμβάνοντας τα κατάλληλα κλειδιά από τις αλυσίδες και θέτοντας τα στις καθολικές μεταβλητές activeKey1 και activeKey2.



Σχήμα 6.3 Διάγραμμα ροής της timeCorrespondingKey

Συμβολισμοί:

difference: Χρονική διαφορά μεταξύ του χρόνου της στιγμής κλήσης της μεθόδου και της τελευταίας χρονικής στιγμής που έγινε αλλαγή του κλειδιού (*curTs*). Η *curTs* ισοδυναμεί με

το δεύτερο άκρο του χρονικού διαστήματος, ενώ η πρώτη είναι η στιγμή κλήσης της μεθόδου (*curTime*). Ακολουθεί ο κώδικας που υλοποιεί το παραπάνω αλγόριθμο.

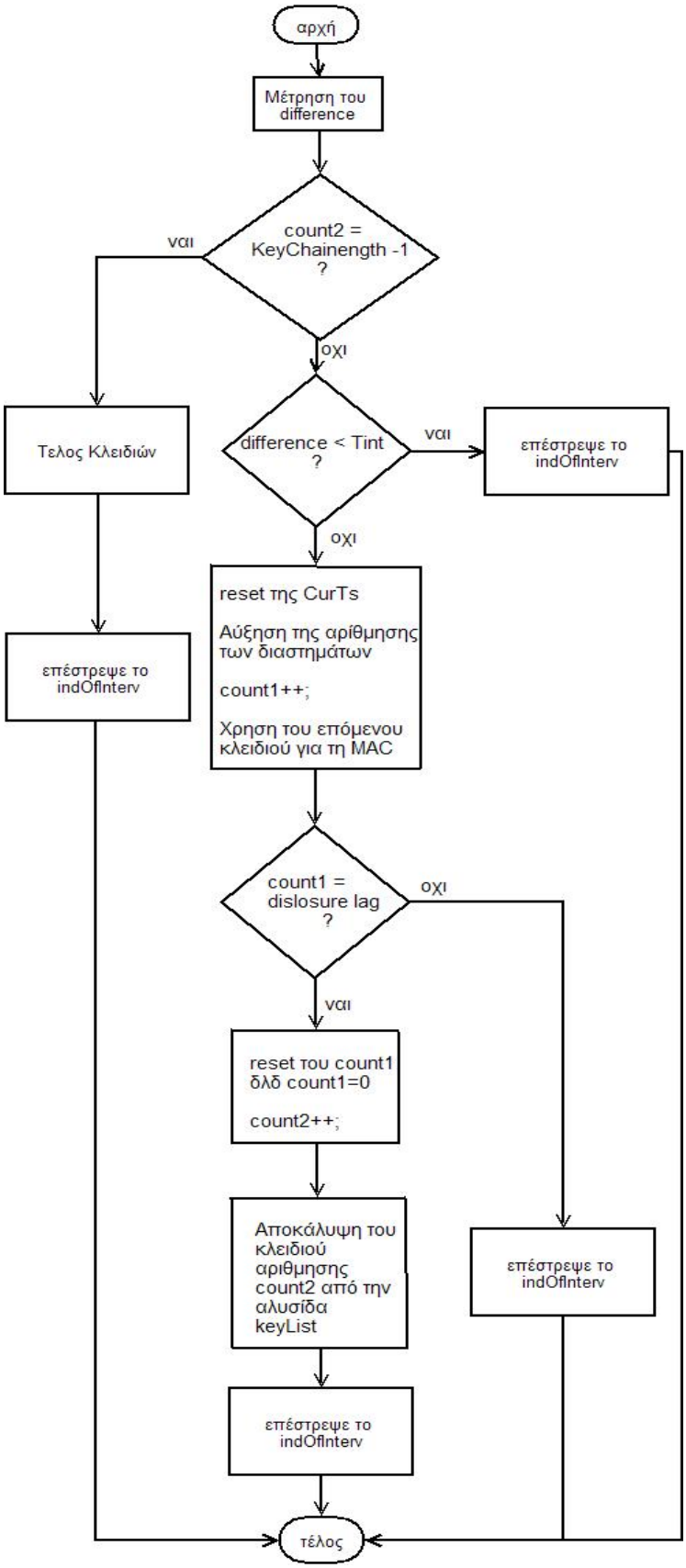
Κώδικας:

```
public void timeCorrespondingKey(ArrayList keyList,ArrayList keyMac,KeyChain keyChain) {
    String currentTime = getTime();
    Double curTime=Double.parseDouble(currentTime);

    boolean timePassed=false;

    Double difference=curTime - this.curTs;

    /**To Tint paristanei ti xroniki diarkeia tou kathe interval
     *Otan perasei xronos=Tint auksanetai o metritis twn distimatwn
     *indOfInterv
     */
    if(this.indOfInterv==(keyChainLength-1)){
        System.out.println(NameThread+":Telos kleidiwn .");
        count2=0;
        this.indOfInterv=0;
    }

    else {
    if(difference<this.Tint)
    {

       timePassed=false;

       this.activeKey2=(Keys)keyMac.get(keyChainLength-this.indOfInterv);


    }
    else{
       System.out.println(NameThread+":difference="+difference);
       this.curTs=curTime; //reset tis arxis metrisis tou diastimatos
```

```
        this.indOfInterv ++; //auksisi tou deikti diastimatwn
        count1++;
        timePassed=true;
        this.activeKey2=(Keys)keyMac.get(keyChainLength-this.indOfInterv);
        if(count1==this.disclosureLag)
        {
          System.out.println(NameThread+":perase i disclosure lag="+this.disclosureLag);
          count1=0;
          count2++;      //o metritis count2 paristanei tin arithmisi tou kleidiou pou tha
                         //apokalifthei
                         //ston client.Auksanetai otan perasei xronos disclosureLag
          System.out.println(NameThread+":count2="+count2);
          this.activeKey1=(Keys)keyList.get(keyChainLength-1-count2); // apokaliptei to
                                                                      //kleidi ston client
       System.out.println(NameThread+":Apokaliptetai to kleidi:"+this.activeKey1.sel+" sto
       interval:"+this.indOfInterv);

        }

    }
    }

}
```

6.2.3 Μέθοδος discloseRestOfKeys

Σκοπός της μεθόδου είναι η αποκάλυψη των κλειδιών που απομένουν στην αλυσίδα, μετά την χρονική στιγμή στην οποία ο αποστολέας διαβάζει το EOF του αρχείου αποστολής δεδομένων. Η λογική είναι η ίδια με την timeCorrespondingKey.Οι συμβολισμοί που χρησιμοποιήθηκαν είναι οι ίδιοι. Παρακάτω δίνεται το διάγραμμα ροής της μεθόδου (σχήμα 6.4) και ο κώδικας που υλοποιεί τον αλγόριθμο.



Σχήμα 6.4:Διάγραμμα ροής της DiscloseRestOfKeys

Κώδικας:

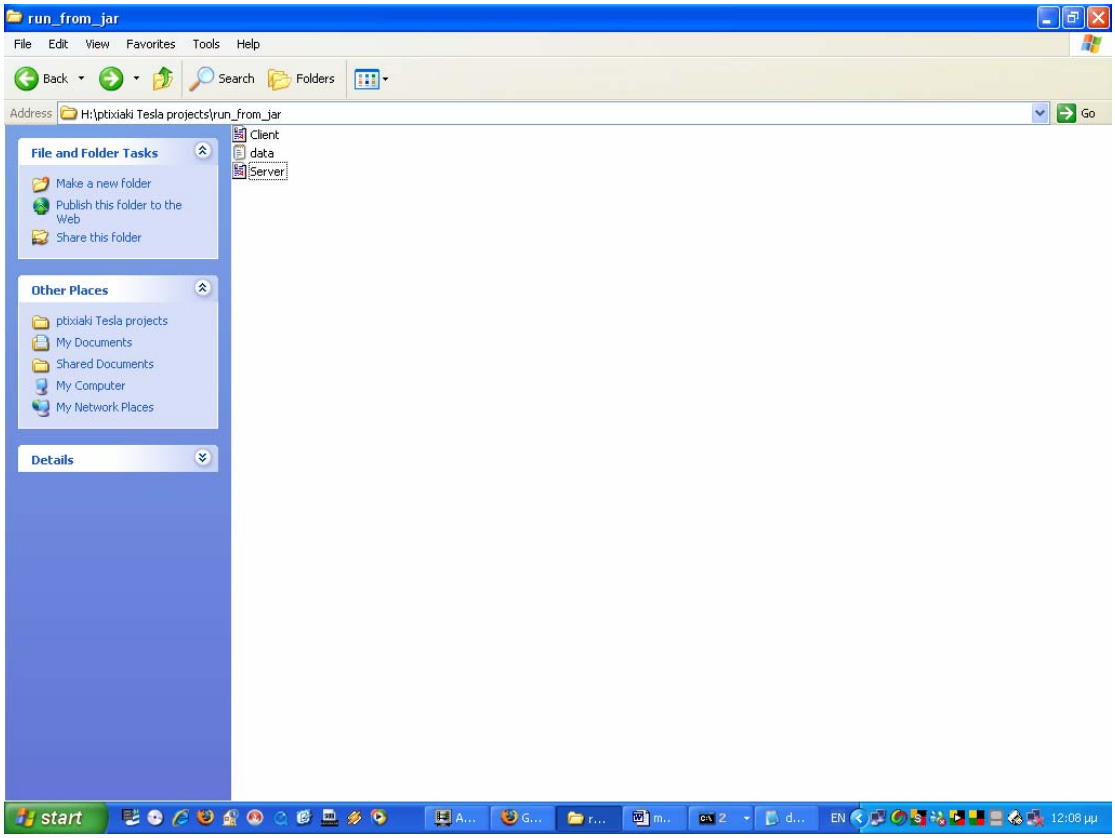
```
public int discloseRestOfKeys(ArrayList keyList) {
        String currentTime = getTime();
        Double curTime=Double.parseDouble(currentTime);
        boolean timePassed=false;

        Double difference=curTime - this.curTs;

        /**To Tint paristanei ti xroniki diarkeia tou kathe interval
         *Otan perasei xronos=Tint auksanetai o metritis twn distimatwn
         *indOfInterv
         */
        if(this.count2==(keyChainLength-1)){
         System.out.println("Telos kleidiwn .");
         }

        else {
        if(difference<this.Tint)
        {

           timePassed=false;

        }
        else{
           System.out.println(NameThread+":difference="+difference);
           this.curTs=curTime; //reset tis arxis metrisis tou diastimatos
           this.indOfInterv ++; //auksisi tou deikti diastimatwn
           this.count1++;
           timePassed=true;
           if(count1==this.disclosureLag)
```

```
        {
            System.out.println(NameThread+":perase i disclosure lag="+this.disclosureLag);
            count1=0;
            count2++;       //o metritis count2 paristanei tin arithmisi tou kleidiou pou tha
                            //apokalifthei
                             //ston client.Auksanetai otan perasei xronos disclosureLag
            System.out.println(NameThread+":count2="+this.count2);
            //apokaliptei to kleidi ston client
            this.activeKey1=(Keys)keyList.get(keyChainLength-1-this.count2);
            System.out.println(NameThread+":Apokaliptetai to kleidi:"+this.activeKey1.sel+"
            sto interval:"+this.indOfInterv);
        }
    }
    }
    return indOfInterv;
}
```

7

Έλεγχος

Στο παρόν κεφάλαιο παραθέτουμε μερικά σενάρια λειτουργίας του προγράμματος. Πρέπει να αναφέρουμε ότι για να λειτουργήσει το σύστημα πρέπει τα υποσυστήματα των client και server να είναι γειτονικά directories καθώς επίσης και τα δεδομένα που θα στείλει ο server στο client (σχήμα 7.1). Από τη μεριά του server παρατηρούμε τη δημιουργία των κλειδιών τα οποία δομούνται σε αλυσίδες, την αποστολή των δεδομένων και τη συγχρονισμένη αποκάλυψη των κλειδιών. Από τη μεριά του client παρατηρούμε την αποστολή του αρχικού πακέτου – αίτησης , την απάντηση του server και τα πακέτα τα οποία καταφθάνουν στη συνέχεια αποκαλύπτοντας τα κλειδιά σύμφωνα με το χρονοδιάγραμμα του server. Στο τέλος βλέπουμε και το αρχείο με τα δεδομένα που δημιουργείται το οποίο περιέχει όλα τα αυθεντικοποιημένα δεδομένα που έστειλε ο server.



Σχήμα 7.1:Αποθήκευση των jar αρχείων

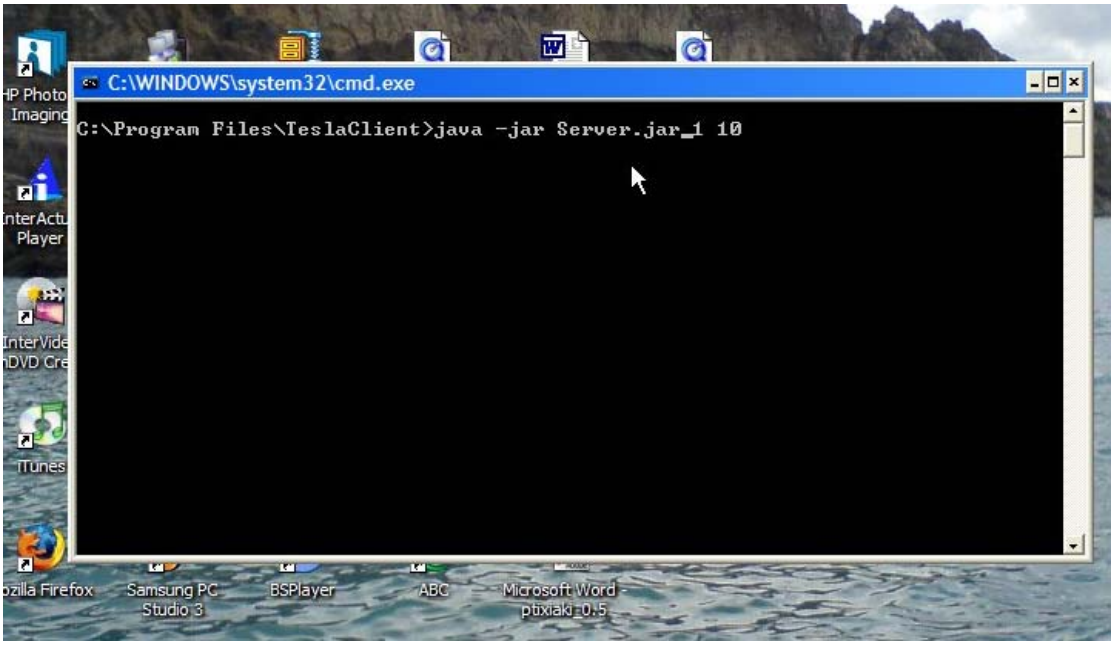
7.1 Μεθοδολογία ελέγχου

7.1.1 Σενάριο «Ένας πελάτης – ένας εξυπηρετητής»

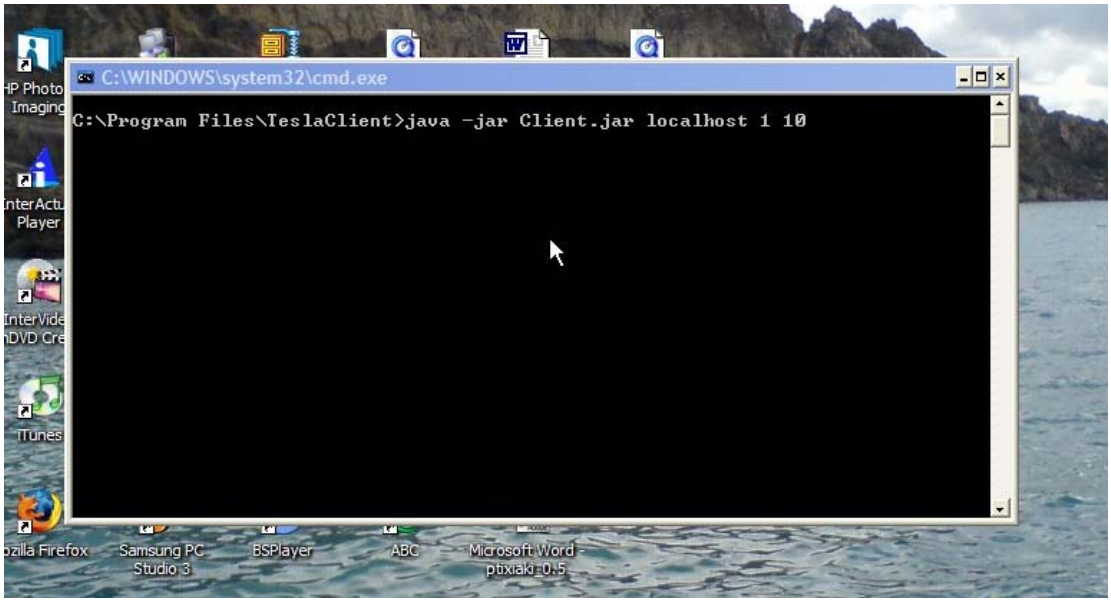
Δίνουμε από command line τις εντολές με την ακόλουθη σειρά. Εκτελούμε πρώτα τον server και μετά τον client:

java –jar Server.jar 1 10

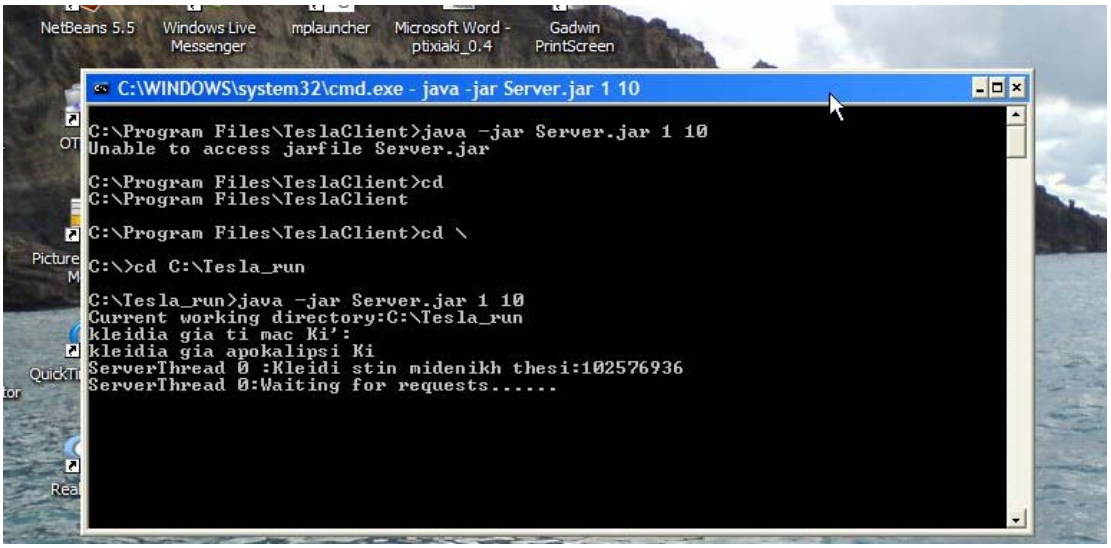
java –jar Client.jar localhost 1 10



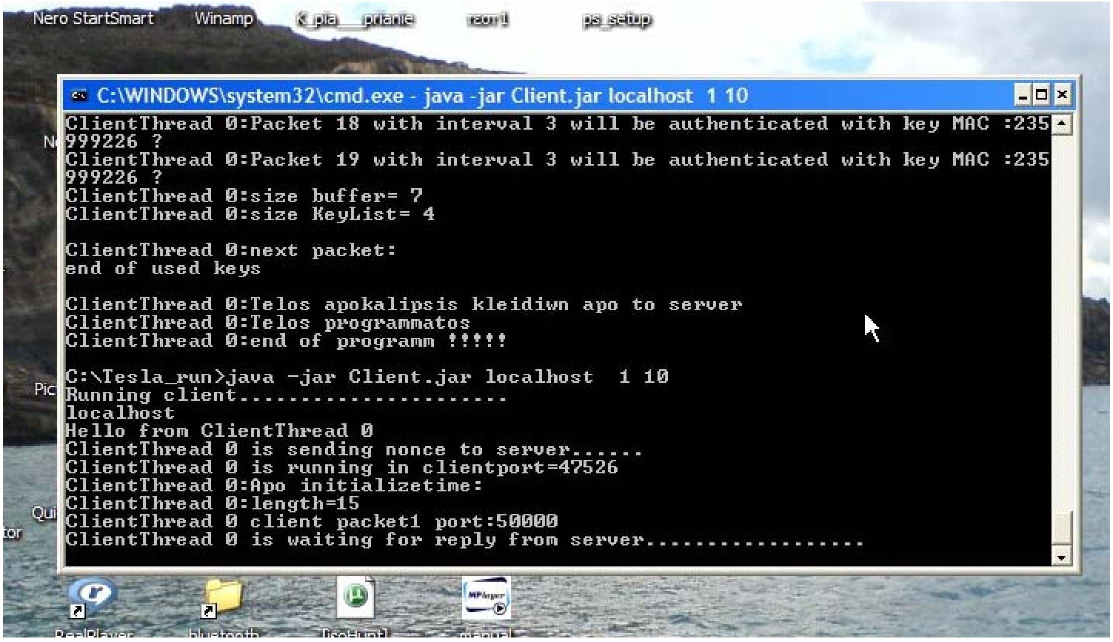
Σχήμα 7.2: Εντολή εκτέλεσης του server



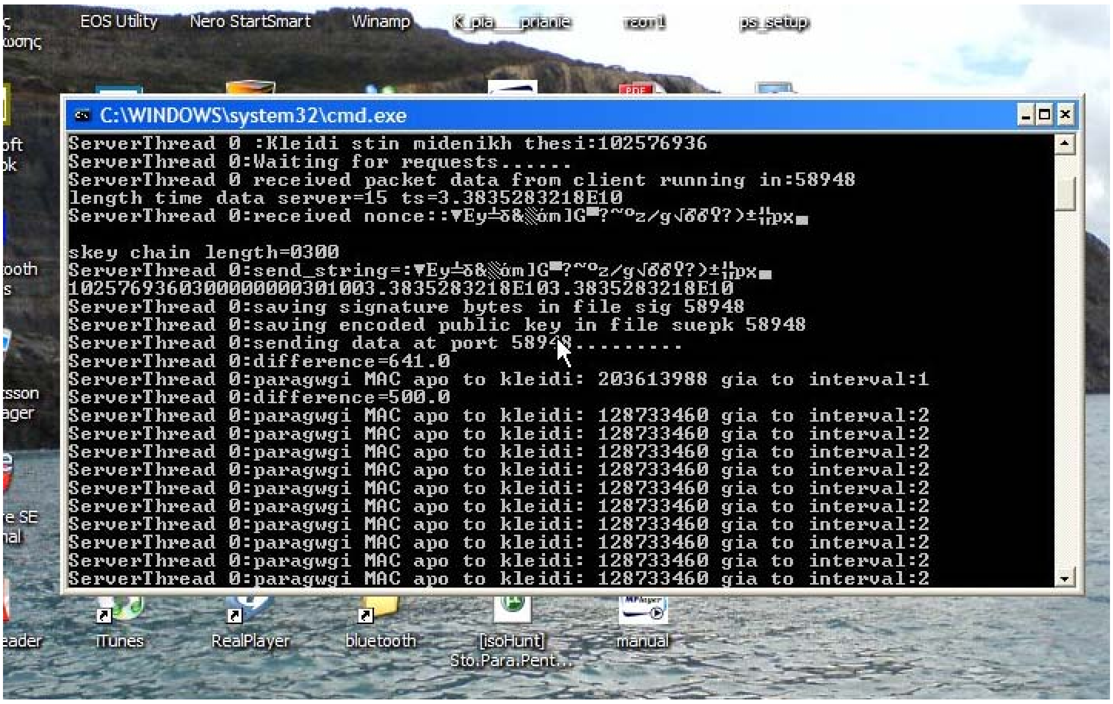
Σχήμα 7.3:Εντολή εκτέλεσης του client



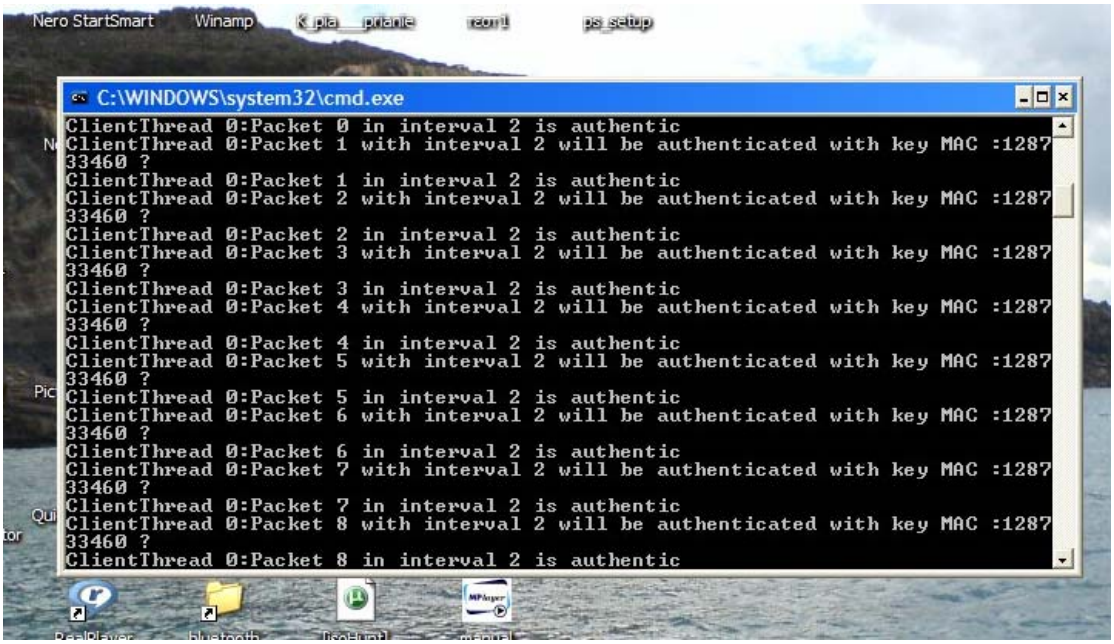
Σχήμα 7.4: Δημιουργία των κλειδιών από το server



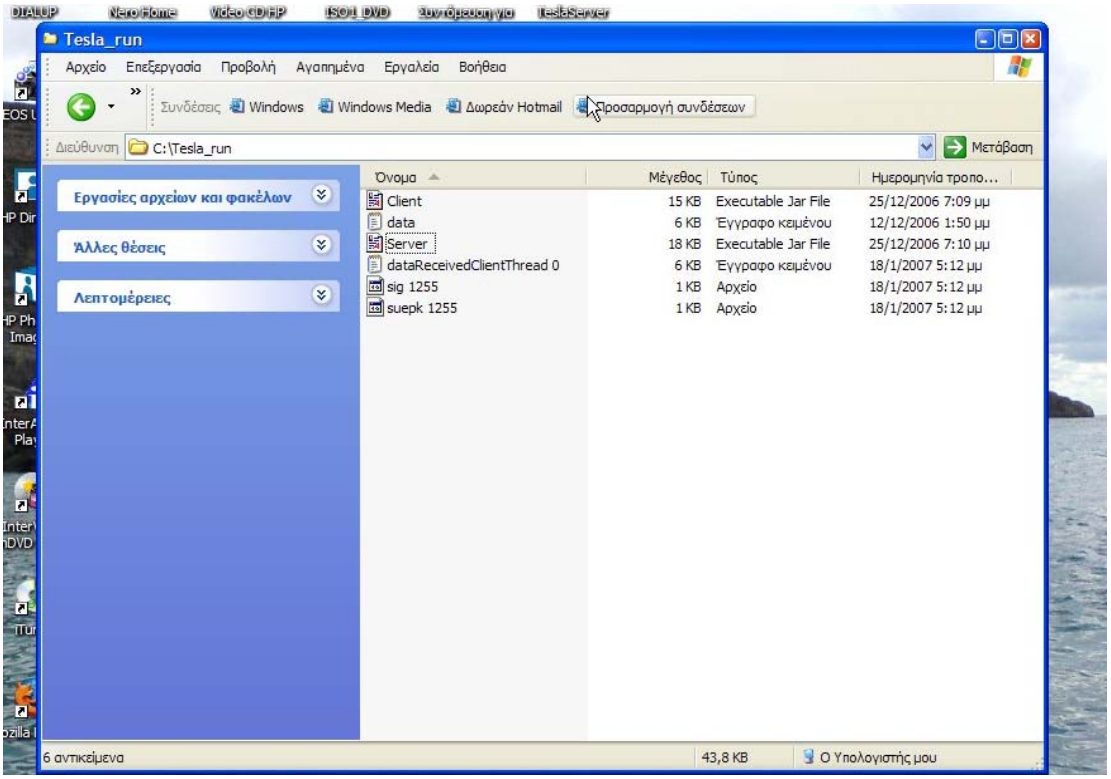
Σχήμα 7.5: Αποστολή αρχικού πακέτου – αίτησης στον server



Σχήμα 7.6:Αποστολή πακέτων και παραγωγή των MAC από το server



Σχήμα 7.7: Λήψη και αυθεντικοποίηση πακέτων από τον client

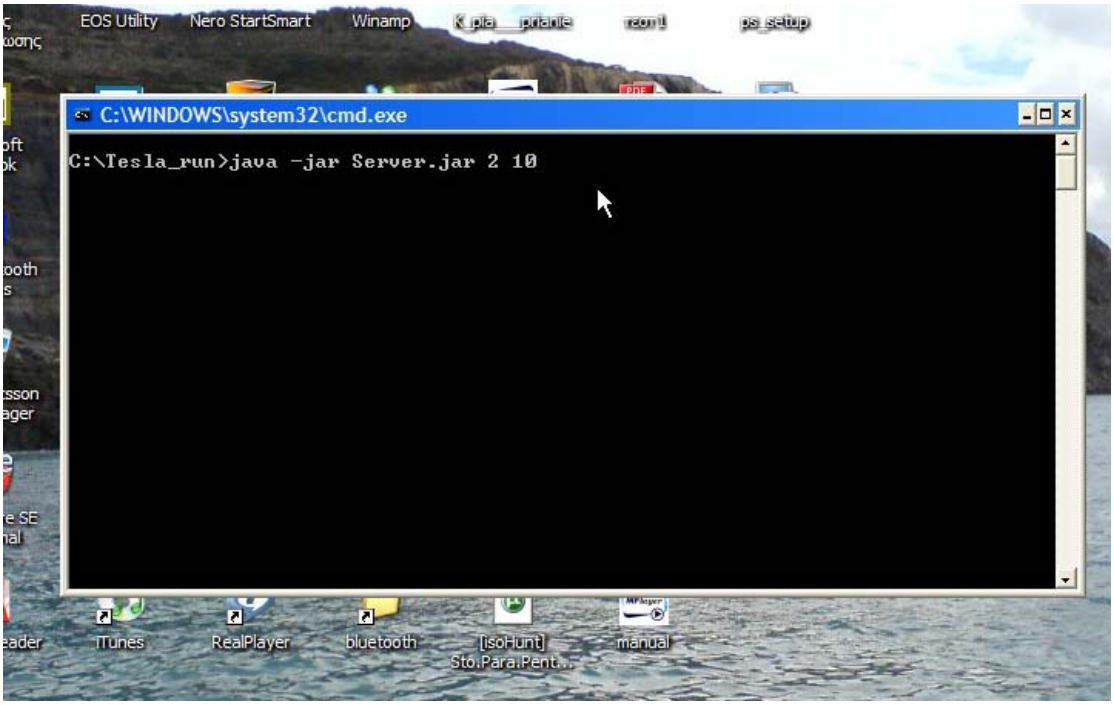


Σχήμα 7.8

Τα δεδομένα λήψης από τον client αποθηκεύονται στο αρχείο dataReceivedClinetThread 0. Ακόμη φαίνονται τα αρχεία όπου αποθηκεύει ο server το δημόσιο κλειδί του και την ψηφιακή υπογραφή των δεδομένων (αρχεία sig 1255, suepk 1255).

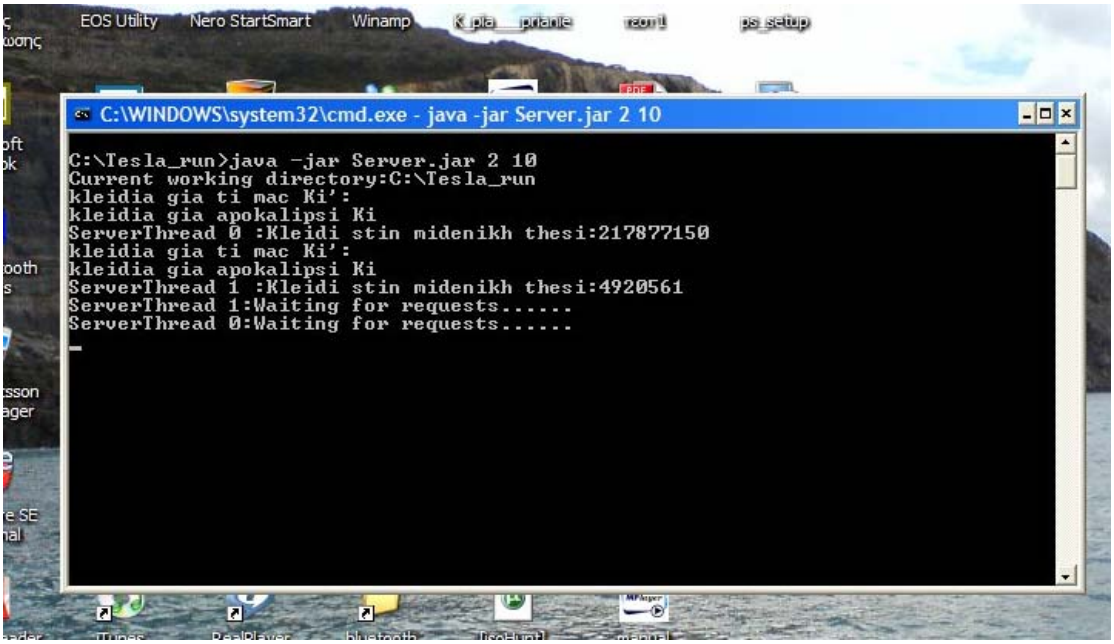
7.1.2 Σενάριο «Δύο πελάτες – δύο εξυπηρετητές»

Σε αυτό το σενάριο λειτουργίας υπάρχουν 2 πελάτες και 2 εξυπηρετητές στο σύστημα. Η λογική είναι παρόμοια με το πρώτο σενάριο λειτουργίας.

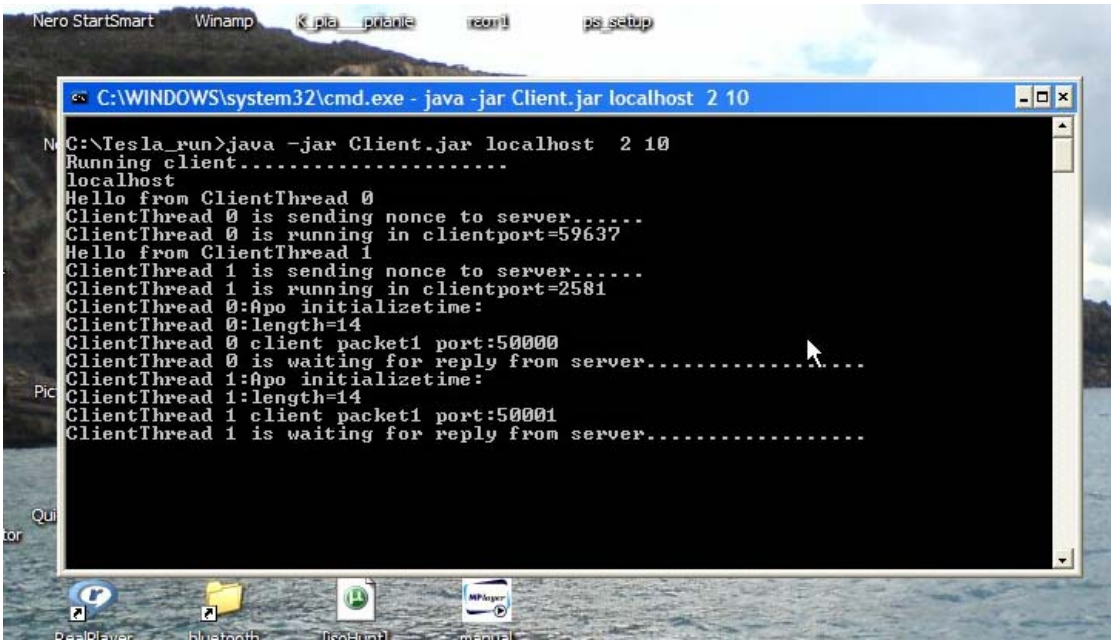


Σχήμα 7.9: Εντολή εκτέλεσης των servers

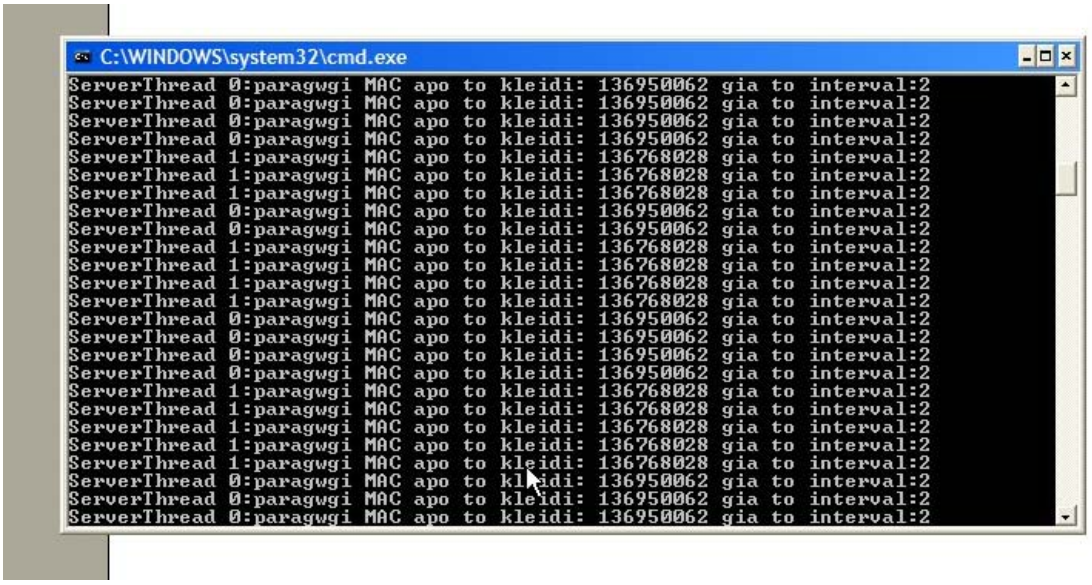
Σχήμα 7.10: Εντολή εκτέλεσης των clients



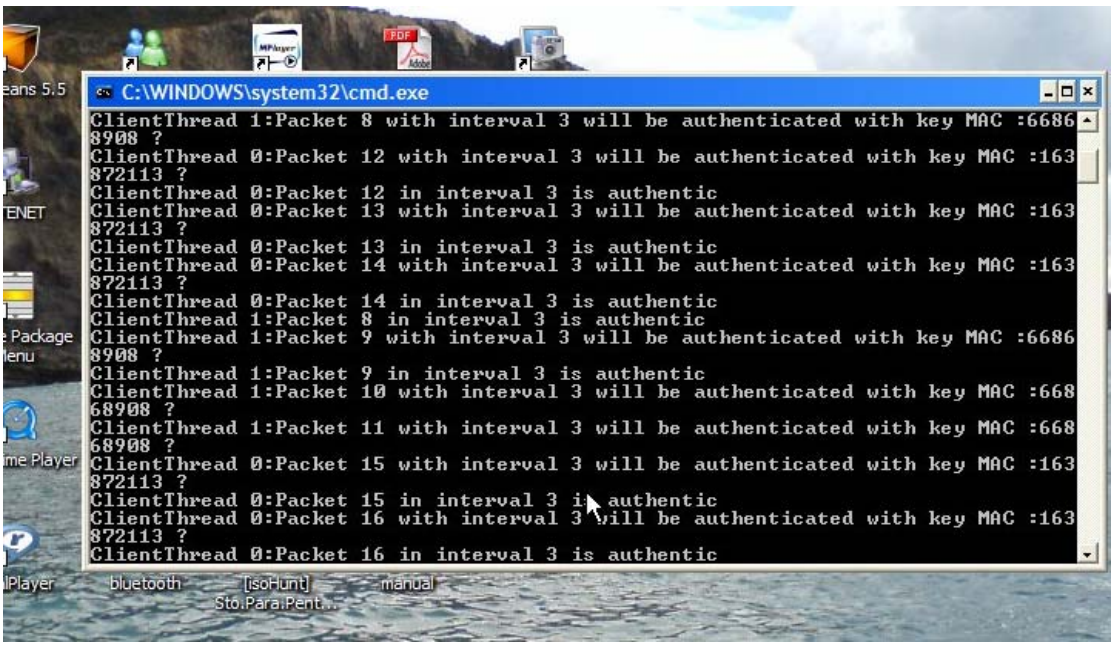
Σχήμα 7.11: Αναμονή για αιτήσεις από τους servers



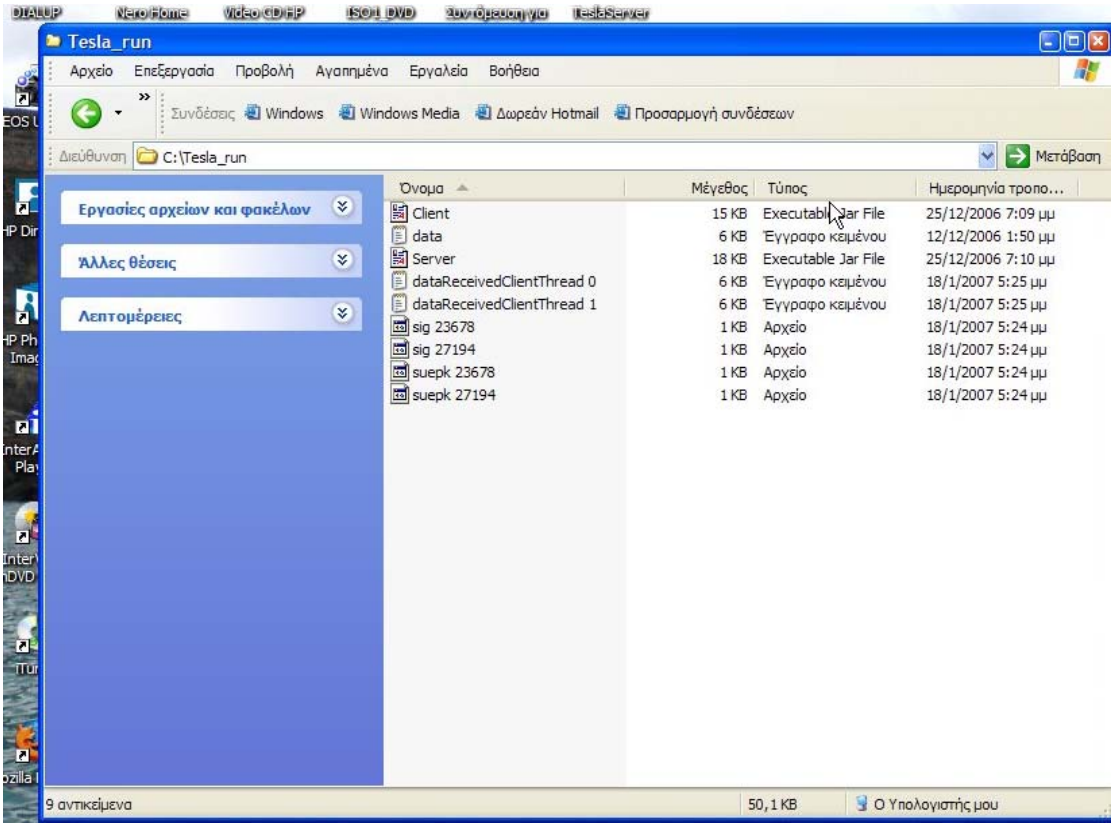
Σχήμα 7.12: Αναμονή των client για απάντηση από τους servers



Σχήμα 7.13: Αποστολή πακέτων από τους servers



Σχήμα 7.14: Λήψη και αυθεντικοποίηση πακέτων από τους clients



Σχήμα 7.15

Τα δεδομένα λήψης από τους clients αποθηκεύονται στα αρχεία dataReceivedClientThread 0 και dataReceivedClientThread 1.Ακόμη φαίνονται τα αρχεία όπου αποθηκεύουν οι servers τα δημόσια κλειδιά τους και τις ψηφιακές υπογραφές των δεδομένων (αρχεία sig 23678,supk 23678, sig 27194,suepk 27194)

8

Επίλογος

Το κεφάλαιο αυτό ολοκληρώνει την παρουσίαση της παρούσας διπλωματικής εργασίας. Στη συνέχεια παρουσιάζεται μία σύντομη ανασκόπησή της και αναφέρεται περιληπτικά μια επέκταση- εφαρμογή του *TESLA* για ασύρματα δίκτυα αισθητήρων, το *μTESLA* .

8.1 Σύνοψη και συμπεράσματα

Όπως αναφέρθηκε στην εισαγωγή οι προκλήσεις και τα θέματα που θέτονται επί τάπητος όσον αφορά τη διάδοση και μετάδοση αυθεντικοποιημένων - ασφαλών πληροφοριών είναι αρκετά και μείζονος σημασίας. Το *TESLA* καταφέρνει να τα ξεπεράσει με μια σχετική εύκολη και απλή προσέγγιση η οποία προσφέρει όμως ισχυρή ασφάλεια και επαλήθευση της ταυτότητας του αποστολέα, για τους χρήστες – παραλήπτες του συστήματος. Κοντολογίς η προσέγγιση του *TESLA* βασίζεται στην γνώση ενός άνω ορίου του χρόνου που βρίσκεται ο αποστολέας και στην καθυστερημένη αποκάλυψη των κλειδιών από τα οποία δημιουργούνται οι *MACs* των πακέτων. Ακόμη κάνει χρήση μονόδρομων αλυσίδων προκειμένου να επαληθεύει ότι τα κλειδιά που αποκαλύπτονται, προέρχονται πράγματι από τον ίδιο αποστολέα και να επαληθεύει έτσι την ταυτότητα του. Το *TESLA* παρόλο που χρησιμοποιεί συμμετρικές έννοιες της κρυπτογραφίας πετυχαίνει μη ασύμμετρες ιδιότητες με την καθυστερημένη αποκάλυψη των κλειδιών. Στην παρούσα διπλωματική έγινε μια παρουσίαση του συγκεκριμένου πρωτοκόλλου και πραγματοποιήθηκε η υλοποίηση αυτού στη γλώσσα *JAVA* όπου φαίνεται ιδίοις όμμασι η ισχυρή ασφάλεια την οποία προσφέρει το *TESLA* για κάποιο χρήστη του συστήματος.

8.2 Επεκτάσεις – εφαρμογές

Το *TESLA* έχει εφαρμοσθεί από τους δημιουργούς του στο πανεπιστήμιο UC Berkeley, στην μετάδοση δεδομένων και επικοινωνία μεταξύ αισθητήρων [38]. Η επικοινωνία επιτυγχάνεται μέσω ασύρματων δικτύων. Στην προκειμένη περίπτωση όμως το *TESLA* δεν μπορεί να εφαρμοσθεί όπως έχει οριστεί μέχρι τώρα λόγω της περιορισμένης επεξεργαστικής ισχύς και της μικρής μνήμης που διαθέτουν οι κόμβοι των δικτύων αισθητήρων. Για αυτούς τους λόγους, έχει δημιουργηθεί το *μTESLA* το οποίο είναι μια παραλλαγή του *TESLA* κατάλληλη

για τα ασύρματα δίκτυα αισθητήρων. Εδώ θα αναφέρουμε επιγραμματικά τα σημεία στα οποία διαφοροποιείται το *μTESLA* από το *TESLA*.

8.2.1 Σύγκριση μTESLA και TESLA

Το βασικό πρωτόκολλο χρησιμοποιεί για την αυθεντικοποίηση του αρχικού πακέτου από τον αποστολέα την έννοια των ψηφιακών υπογραφών. Όμως αυτό δεν μπορεί να επιτευχθεί σε δίκτυα αισθητήρων λόγω της μικρής επεξεργαστικής ισχύς τους. Χαρακτηριστικά αναφέρουμε ότι στη μνήμη των αισθητήρων του δικτύου δεν θα χωρούσε ούτε ο απαραίτητος κώδικας των ψηφιακών υπογραφών. Ακόμη το μέγεθος των πακέτων του *TESLA* είναι περίπου 24 bytes ανά πακέτο. Για δίκτυα τα οποία ενώνουν τερματικούς σταθμούς αυτό δεν είναι σημαντικό. Όμως οι κόμβοι - αισθητήρες στέλνουν κατά προσέγγιση μηνύματα μεγέθους 30 bytes. Είναι απλά μη πρακτικό να αποκαλύπτει κάθε πακέτο το κλειδί που χρησιμοποιήθηκε για τη *MAC* σε κάθε πακέτο. Με κλειδιά μήκους 64 bits και *MACs* το σχετιζόμενο με το *TESLA* κομμάτι θα αποτελούσε το 50% του κάθε πακέτου. Τέλος, οι μονόδρομες αλυσίδες δεν χωράνε στη μνήμη ενός κόμβου – αισθητήρα.

Το *μTESLA* σχεδιάστηκε ειδικά για τα ασύρματα δίκτυα αισθητήρων με τις εξής διαφοροποιήσεις:

- Το *μTESLA* δεν χρησιμοποιεί για την αυθεντικοποίηση του πρώτου πακέτου συγχρονισμού ψηφιακές υπογραφές αλλά μόνο συμμετρικές ιδιότητες. Η επαλήθευση γίνεται μέσω της nonce που στέλνει ο παραλήπτης.
- Το *μTESLA* δεν αποκαλύπτει σε κάθε πακέτο τα κλειδιά των αντίστοιχων διαστημάτων, αλλά προκειμένου να εξοικονομήσει ενέργεια στέλνει περιοδικά ειδικά πακέτα που περιέχουν τα συγκεκριμένα κλειδιά.
- Λόγω ανεπαρκής μνήμης των κόμβων – αισθητήρων το *μTESLA* περιορίζει τον αριθμό των αποστολέων στο σύστημα.

Συνοψίζοντας βλέπουμε ότι το *μTESLA* μπορεί να εφαρμοστεί επιτυχώς στα ασύρματα δίκτυα αισθητήρων προσφέροντας ένα ισχυρό πυρήνα ασφάλειας για τους χρήστες του συστήματος.

9

Βιβλιογραφία και references links:

[1] Ross J. Anderson, Francesco Bergadano, Bruno Crispo, Jong-Hyeon Lee, Charalampos Manifavas, and Roger M. Needham. A new family of authentication protocols. *Operating Systems Review*, 32(4):9–20, October 1998.

[2] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In Yvo Desmedt, editor, *Advances in Cryptology - Crypto '94*, pages 341–358,Berlin, 1994. Springer-Verlag. Lecture Notes in Computer Science Volume 839.

[3] M. Bellare and P. Rogaway. Collision-resistant hashing:Towards making UOWHFs practical. In Burt Kaliski, editor, *Advances in Cryptology - Crypto '97*,pages 470–484, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1294.

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Message Authentication using Hash Functions—The HMAC Construction. *RSA Laboratories CryptoBytes*,2(1), Spring 1996.

[5] Matt Bishop. A Security Analysis of the NTP Protocol Version 2. In *Sixth Annual Computer Security Applications Conference*, November 1990.

[6] M. Borella, D. Swider, S. Uludag, and G. Brewster. Internet packet loss: Measurement and implications for end-to-end qos. In *International Conference on Parallel Processing*, August 1998.

[7] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security:A taxonomy and some efficient constructions. In *Infocom '99*, 1999.

[8] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *JCSS No. 18*, (18):143–154, 1979.

[9] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM symposium on Communications architectures and protocols SIGCOMM '90*, pages 200–208, September 26-28 1990.

[10] Cryptix. `http://www.cryptix.org`.

[11] Stephen E. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proceedings of ACMSIGCOMM '88*, August 1988.

[12] T. Dierks and C. Allen. The TLS protocol version 1.0.Internet Request for comments RFC 2246, January 1999. Proposed standard.

[13] Rosario Gennaro and Pankaj Rohatgi. How to Sign Digital Streams. Technical report, IBM T.J.Watson Research Center, 1997.

[14] Oded Goldreich. Foundations of cryptography (fragments of a book). `http://www.toc.lcs.mit.edu/\~oded/frag.html`, 1998.

[15] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen message attacks. *SIAM Journal of Computing*,17(2):281–308, April 1988.

[16] Mark Handley. Private communication with Adrian Perrig, February 2000.

[17] IBM. Java web page. `http://www.ibm.com/developer/java`.

[18] Ipsec. IP Security Protocol, IETF working group. `http://www.ietf.org/html.charters/ipsec-charter.html`.

[19] Michael George Luby. *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes, 1996.

[20] R. C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology - Crypto '89*, pages 218–238, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 435.

[21] Ralph Merkle. Protocols for public key cryptosystems.In *1980 IEEE Symposium on Security and Privacy*,1980.

[22] David L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. Internet Request for Comments, March 1992. RFC 1305.

[23] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing (STOC '89)*, 1989.

[24] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[25] M. O. Rabin. The information dispersal algorithm and its applications, 1990.

[26] Ronald L. Rivest. The MD5 message-digest algorithm.Internet Request for Comments, April 1992.RFC 1321.

[27] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman.A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1(2):120–126, 1978.

[28] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *6th ACM Conference on Computer and Communications Security*, November 1999.

[29] Secure Multicast User Group (SMUG). `http://www.ipmulticast.com/community/smug/`.

[30] Paul F. Syverson, Stuart G. Stubblebine, and David M.Goldschlag. Unlinkable serial transactions. In *Financial Cryptography '97, Springer Verlag, LNCS 1318*,1997.

[31] C. K.Wong and S. S. Lam. Digital signatures for flows and multicasts. In *Proc. IEEE ICNP '98*, 1998.

[32] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modelling of the temporal dependence in packet loss. In *IEEE INFOCOM '99*, New York,NY, March 1999.

[33]Cisco Multicast Routing & Switching (Κεφάλαια 1 & 2).

[34]A. Perrig, R. Canetti, J. D. Tygar, and D. Song. Effcient authentication and Signing of multicast streams over Lossy Channels. In Proceedings of the IEEE Symposium on Research in Security and Privacy, pages 56–73, May 2000.

[35]The *TESLA* Broadcast Authentication Protocol, Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Song.

[36]http://www.faqs.org/faqs/client-server-faq/

[37] http://www.codeproject.com/threads/jek.asp

[38] SPINS: Security Protocols for Sensor Networks, Adrian Perrig ,Robert Szewczyk, Victor Wen and David E.Culler

[39] http://noc.auth.gr/services/general/rootca/cryptography/