# Ανοικτό Πανεπιστήμιο Κύπρου

**Σχολή Θετικών και Εφαρμοσμένων Επιστημών**

**Μεταπτυχιακό Πρόγραμμα Σπουδών *Πληροφοριακά και Επικοινωνιακά Συστήματα***

# Μεταπτυχιακή Διατριβή

**Επιθέσεις σε πρωτόκολλα ασφάλεια διαδικτύου (IPSEC, TLS)**

**Χριστάκης Ιωάννου**

**Επιβλέπων Καθηγητής**
**Νικόλαος Ε. Κολοκοτρώνης**

**Μάιος 2016**

Ανοικτό Πανεπιστήμιο Κύπρου

**Σχολή Θετικών και Εφαρμοσμένων Επιστημών**

**Μεταπτυχιακό Πρόγραμμα Σπουδών *Πληροφοριακά και Επικοινωνιακά Συστήματα***

# Μεταπτυχιακή Διατριβή

## Επιθέσεις σε πρωτόκολλα ασφάλεια διαδικτύου (IPSEC, TLS)

**Χριστάκης Ιωάννου**

**Επιβλέπων Καθηγητής**
**Νικόλαος Ε. Κολοκοτρώνης**

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων για απόκτηση μεταπτυχιακού τίτλου σπουδών στα Πληροφοριακά και Επικοινωνιακά Συστήματα από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών του Ανοικτού Πανεπιστημίου Κύπρου.

**Μάιος 2016**

## Περίληψη

Η ολοένα αυξανόμενη χρήση του διαδικτύου και η επέκταση των δραστηριοτήτων του, που περιλαμβάνουν πληροφόρηση, ανταλλαγή ηλεκτρονικών μηνυμάτων (email), πληρωμές λογαριασμών και υπηρεσιών, ηλεκτρονικό εμπόριο και τραπεζικές συναλλαγές, εγείρουν σοβαρά ζητήματα όσον αφορά την ασφάλεια στο διαδίκτυο. Η εξασφάλιση της ιδιωτικότητας της προσωπικής ζωής και η διαφύλαξη των περιουσιακών στοιχείων αποτελεί μείζων ζήτημα και πετυχαίνετε μέσω των πρωτοκόλλων ασφαλείας (IPsec, TLS/SSL). Τα πρωτόκολλα ασφαλείας χαρακτηρίζονται από δυο φάσεις, τη φάση της χειραψίας και της ανταλλαγής δεδομένων.

Σκοπός της Μεταπτυχιακής Διατριβής είναι η ανάδειξη των αδυναμιών και των ευπαθειών που παρουσιάζουν τα σημερινά πρωτόκολλα ασφαλείας στο διαδίκτυο, με στόχο την ενημέρωση, αλλά και ευαισθητοποίηση του κοινού γύρω από την ασφάλεια στο διαδίκτυο, την κατανόηση των αιτιών που προκαλούν τις αδυναμίες αυτές, καθώς και τον τρόπο αποφυγής τους.

Οι αδυναμίες αυτές οδηγούν σε μια πολύ γνωστή ομάδα επιθέσεων, τις επιθέσεις ενδιάμεσου ατόμου (Man-In-The-Middle Attack(MITM Attack)). Ο επιτιθέμενος προσπαθεί να μπει στο ενδιάμεσο της σύνδεσης μεταξύ πελάτη και εξυπηρετητή, και εκμεταλλευόμενος τις αδυναμίες του πρωτοκόλλου ασφαλείας ή του αλγόριθμου ανταλλαγής κλειδιών κατά τη φάση της χειραψίας, να υποκλέψει τα κλειδιά της σύνδεσης, να καταλάβει τη σύνδεση, και να παρακολουθεί την ανταλλαγή δεδομένων.

Η Μεταπτυχιακή Διατριβή, πραγματώνεται τις επιθέσεις στα πρωτόκολλα διαδικτύου. Έτσι για το σκοπό αυτό, δημιουργήθηκε ένα πειραματικό περιβάλλον για τη δημιουργία και εκτέλεση ελεγχόμενων επιθέσεων. Για την επίτευξη του παραπάνω, χρειάστηκε η δημιουργία ενός Apache εξυπηρετητή και εγκατάσταση του εργαλείου FlexTLS σε ένα άλλο υπολογιστή. Το FlexTLS είναι ένα εργαλείο γραμμένο σε γλώσσα προγραμματισμού F# και βασίζεται στη βιβλιοθήκη miTLS, μια επικυρωμένη εφαρμογή του TLS. Το FlexTLS αναπτύχθηκε με σκοπό τη δημιουργία επιθέσεων και τον έλεγχο της καινούργιας έκδοσης του πρωτοκόλλου TLS v1.3. Πραγματοποιήθηκαν τρεις διαφορετικές και επιτυχημένες επιθέσεις ενδιαμέσου που οφείλονταν σε ευπάθειες του πρωτοκόλλου ασφαλείας και της κρυπτογραφικής βιβλιοθήκης OpenSSL. Επίσης αναλύθηκαν και αναπτύχθηκαν δυο μεγάλες επιθέσεις που οδήγησαν στην κατάρρευση του TLS v1.2 η FREAK και η Logjam.

## Summary

In our days the web usage is been dramatically increase, as a lot of activities in the internet, like information, email, paying bills, e-commerce and bank transactions. So web security is major issue, and the safeguard of privacy and property is being achievement over security protocols (IPsec, TLS/SSL). The security protocols are completed in two phases, the handshake phase and the data exchange phase.

The Master Thesis aim is to show the vulnerabilities and exposures at the networking security protocols. Web security is something that everybody must know and be awareness of, to know why vulnerabilities presents and how we can avoid them.

The most known cyber-attack is man in the middle attacks. Man in the middle attempts to hijack a connection, intercepts every package is being transmitted between client and server and read the package content or even tamper it. This is achievement over vulnerabilities are presents on security protocols or key exchange algorithm through negotiation progress.

The main purpose of the Master Thesis is the implementation of man in the middle attacks. So, I create an experimental environment to create and implement cyber-attacks. First of all, I need to create an Apache web server, and then to install on another computer the FlexTLS software. The FlexTLS is written in F# functional programming language and it reuse messaging and cryptographic modules of miTLS, a verified reference implementation of TLS. The FlexTLS is develop in order to implement man in the middle attacks and to examine and test TLS v1.3. In this Thesis three successfully man in the middle attacks were achievement and described. This attacks are owed to vulnerabilities and exposures of TLS protocol and OpenSSL cryptographic library. Two of the biggest cyber-attacks, FREAK and Logjam been analysed and descripted in the paper, are responsible for the TLS v1.2 collapse and the design of TLS v1.3 new protocol.

## Ευχαριστίες

Εδώ θα ήθελα να ευχαριστήσω τον καθηγητή μου και καθοδηγητή μου κύριο Κολοκοτρώνη για την πολύτιμη βοήθεια που μου πρόσφερε για να φέρω εις πέρα τη Μεταπτυχιακή Διατριβή. Θα ήθελα να τον ευχαριστήσω για το χρόνο που ξόδεψε για να μου μεταλαμπαδέψει τις γνώσεις του και να απαντήσει στις απορίες μου. Επίσης, για την υπομονή και την επιμονή που επέδειξε, ώστε να καταφέρουμε να έχουμε τα επιθυμητά αποτελέσματα στην εργασία.

Τέλος θα ήθελα να ευχαριστήσω τόσο την οικογένεια μου, όσο και τους φίλους μου, που μου στάθηκαν σ' αυτό το δύσκολο έργο, αλλά και για την υπομονή που έδειξαν τόσο καιρό.

# Περιεχόμενα

Περίληψη ..... iii
Summary ..... iv
Ευχαριστίες ..... v
Περιεχόμενα ..... vi

**Κεφάλαιο 1 Εισαγωγή ..... 1**
1.1 Το Πρόβλημα της Ασφάλειας στο Διαδίκτυο ..... 2
1.2 Πως Αντιμετωπίζεται η Ασφάλεια στο Διαδίκτυο ..... 3
1.3 Πως Επιτυγχάνεται η Ασφάλεια στο Διαδίκτυο ..... 5

**Κεφάλαιο 2 Κρυπτογραφία ..... 7**
2.1 Τυχαίοι και Ψευδοτυχαίοι Αριθμοί ..... 8
2.2 Ρυθμός Λειτουργίας CBC (Cipher Block Chaining) ..... 9
2.3 Συμμετρική Κρυπτογραφία ..... 11
2.3.1 DES (Data Encryption Standard) ..... 11
2.3.2 TDES ή 3DES (Triple Data Encryption Standard) ..... 12
2.3.3 AES (Advance Encryption Standard) ..... 13
2.4 Ασύμμετρη Κρυπτογραφία ..... 14
2.4.1 Αλγόριθμος RSA ..... 16
2.4.2 Αλγόριθμος ανταλλαγής κλειδιού Diffie-Hellman ..... 18
2.5 Αυθεντικοποίηση ..... 20
2.5.1 Κώδικας Αυθεντικοποίησης Μηνυμάτων (MAC) ..... 20
2.5.2 Ψηφιακή Υπογραφή (Digital Signature) ..... 21
2.6 Ψηφιακά Πιστοποιητικά (Digital Certificates) ..... 22
2.6.1 Το Πιστοποιητικό X.509 ..... 24

**Κεφάλαιο 3 Πρωτόκολλα Ασφαλείας ..... 25**
3.1 Secure Socket Layer (SSL) ..... 26
3.1.1 SSL Record Protocol ..... 29
3.1.2 Change Cipher Spec Protocol ..... 30
3.1.3 Alert Protocol ..... 30

3.1.4 Handshake Protocol ..... 31
3.2 Transport Layer Security (TLS) ..... 38
3.3 IPsec ..... 42
3.3.1 Το Πρωτόκολλο Authentication Header (AH) ..... 44
3.3.2 Το Πρωτόκολλο Encapsulating Security Payload (ESP) ..... 45
3.3.3 Transport and Tunnel Mode ..... 46
3.3.4 Το πρωτόκολλο Internet Key Exchange (IKE) ..... 48

**Κεφάλαιο 4 Επιθέσεις Ενδιάμεσου ..... 51**
4.1 Επίθεση Κατακερματισμένου Μηνύματος Προειδοποίησης (Fragmented Alert Attack) ..... 55
4.2 Επίθεση Κατακερματισμένου Client Hello Μηνύματος (Fragmented Client Hello Attack) ..... 56
4.3 Επίθεση Παράληψης Μηνύματος (Skip Attack) ..... 57
4.4 Επίθεση EarlyCCS ..... 59
4.5 Επίθεση Τριπλής Χειραψίας (Triple Handshake Attack) ..... 61
4.6 Επίθεση FREAK ..... 66
4.7 Επίθεση Logjam ..... 67

**Κεφάλαιο 5 Λογισμικά Εργαλεία ..... 74**
5.1 OpenSSL ..... 74
5.1.1 Προηγούμενες Εκδόσεις και Ευπάθειες ..... 75
5.1.2 Δημιουργία Ψηφιακών Πιστοποιητικών X.509 ..... 77
5.2 FlexTLS ..... 81
5.3 miTLS ..... 84

**Κεφάλαιο 6 Μοντελοποίηση των Επιθέσεων ..... 85**
6.1 Το Περιβάλλον Εργασίας ..... 85
6.2 Έλεγχος και Υλοποίηση Επιθέσεων ..... 90
6.3 Σύγκριση Αποτελεσμάτων και Εξαγωγή Συμπερασμάτων ..... 100

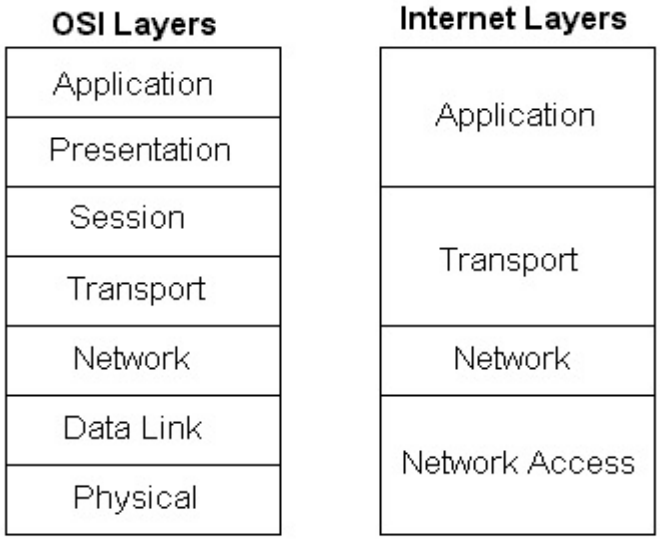
**Κεφάλαιο 7 Επίλογος ..... 104**
Βιβλιογραφία ..... 106

# Κεφάλαιο 1

## Εισαγωγή

Η επικοινωνία και η ανταλλαγή δεδομένων, αποτελούν θεμελιώδεις λειτουργίες των πληροφορικών και επικοινωνιακών συστημάτων. Το διαδίκτυο (Internet), είναι ένα τρανταχτό παράδειγμα πληροφόρησης και ανταλλαγής δεδομένων μεταξύ χρηστών (clients) και εξυπηρετητών (servers). Το διαδίκτυο αποτελεί σήμερα ένα αναπόσπαστο κομμάτι της ζωής μας και της καθημερινότητάς μας. Όλο και περισσότεροι άνθρωποι, αναζητούν, διαμοιράζονται, και χρησιμοποιούν τις προσφερόμενες υπηρεσίες, από τουλάχιστόν μια συσκευή (Βογιατζάκης, Α., Γ., 2008, σσ. 13-14) (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007, σσ. 21-22).

Το διαδίκτυο βασίζεται πάνω στο πρωτόκολλο επικοινωνίας TCP/IP, το οποίο μοιάζει με το πρωτόκολλο επικοινωνίας OSI (Open System Interconnection) με τη διαφορά, ότι έχει λιγότερα επίπεδα σε σχέση με το OSI. Συγκεκριμένα το πρωτόκολλο OSI περιέχει επτά επίπεδα (φυσικό, σύνδεσης δεδομένων, δικτύου, μεταφοράς, συνόδου, παρουσίασης και εφαρμογής), ενώ το TCP/IP έχει τέσσερα συγχωνεύοντας το επίπεδα φυσικό και σύνδεσης δεδομένων στο επίπεδο σύνδεσης δεδομένων, τα επίπεδα μεταφοράς και συνόδου στο επίπεδο μεταφοράς και τα επίπεδα παρουσίασης και εφαρμογής στο επίπεδο της εφαρμογής, όπως φαίνεται και στην εικόνα 1. Παρόλα αυτά, το TCP/IP εγγυάται τη διακίνηση και τη μετάδοση των δεδομένων ανάμεσα στους κόμβους του διαδικτύου, αλλά όχι την ασφαλή παράδοσή τους (Νούκας, Κ., Παππής, Χ., 2006, σσ. 10-15) (Μάντζιος, Ι., Μπαντίας, Σ., 2007, σσ. 6-8).




**Εικόνα 1.** Τα επίπεδα που αντιπροσωπεύουν τα πρωτόκολλα OSI και TCP/IP (Μάντζιος, Ι., Μπαντίας, Σ., 2007).

## 1.1 Το Πρόβλημα της Ασφάλειας στο Διαδίκτυο

Λόγω της δομής του, το διαδίκτυο αποτελεί ένα κύριο "χώρο" ανταλλαγής πληροφοριών. Συνεπώς όλο και πληθαίνουν οι χρήστες του διαδικτύου, κάτι που οδηγεί σε αντίστοιχη αύξηση των πληροφοριών που ανταλλάσσονται. Όμως κάποιες από τις εν λόγω πληροφορίες μπορεί να είναι ευαίσθητες, ή κρίσιμές, ή ιδιωτικές (όπως αριθμοί πιστωτικών καρτών, κρατικά μυστικά έγγραφα, ιατρικό ιστορικό ασθενούς) και δε θέλουμε να είναι ορατές και προσβάσιμες από τρίτα, μη εξουσιοδοτημένα άτομα (Μάντζιος, Ι., Μπαντίας, Σ., 2007, σσ. 8-9).

Οι κυριότεροι λόγοι που μπορούμε να νιώθουμε ανασφαλείς στο διαδίκτυο είναι πρώτα από όλα ο φόβος της παρακολούθησης (ωτακουστής — eavesdropper) των μηνυμάτων που ανταλλάσσονται μεταξύ υπολογιστή-εξυπηρετητή. Ακόμα ένας λόγος είναι ο κίνδυνος της αλλοίωσης δεδομένων. Τελευταίος λόγος ανασφάλειας είναι η πλαστοπροσωπία, δηλ. και η προσποίηση ότι προσπάθεια κάποιος κάποιου κακόβουλου χρήστης να υποδυθεί έναν είναι κάποιος άλλο χρήστης (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007, σ. 22).

Ο τρόπος που κάποιος κακόβουλος χρήστης μπορεί να επιτύχει τα παραπάνω είναι μέσω των επιθέσεων. Οι επιθέσεις χωρίζονται σε δυο είδη ανάλογα με τον σκοπό του επιτιθέμενου. Το πρώτο είδος είναι οι παθητικές επιθέσεις, που χρησιμοποιούνται κατά

κόρον από τους ωτακουστές, για παρακολουθήσεις. Είναι δύσκολα ανιχνεύσιμες, καθώς ο χρήστης δεν παρατηρεί κάποια περίεργη μεταβολή ή αλλοίωση στην επικοινωνία του. Τα βασικά δυο είδη των παθητικών επιθέσεων είναι η ανάγνωση των περιεχομένων των μηνυμάτων και η ανάλυση κίνησης (traffic analysis). Στην τελευταία περίπτωση, ο επιτιθέμενος, μπορεί να μην είναι εφικτό να διαβάσει τα περιεχόμενα των μηνυμάτων, αλλά να μπορεί να εξάγει διάφορες πληροφορίες από αυτά, όπως την προέλευση των μηνυμάτων, τον προορισμό τους, και τη συχνότητα αποστολής τους, πληροφορίες που μπορεί να αξιοποιήσει και να του φανούν χρήσιμες.

Το δεύτερο είδος επιθέσεων είναι οι ενεργητικές επιθέσεις οι οποίες έχουν ως αποτέλεσμα την τροποποίηση μηνυμάτων. Οι επιθέσεις αυτές χωρίζονται σε πέντε κύρια είδη (Stallings, W., 2010, σσ. 9-13) (Stallings, W., 2014, pp. 15-17) (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007, σσ. 22-23) (Κατσίδου, Μ., 2007, σσ. 17-27).

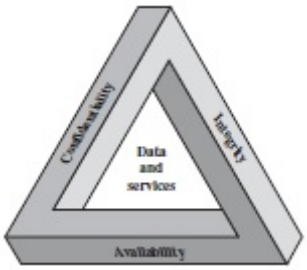
- Το μασκάρεμα (masquerade), το οποίο δεν εκτελείται ποτέ μόνο, αλλά σε συνδυασμό με τις άλλες επιθέσεις. Ο επιτιθέμενος υποδύεται κάποιον άλλον προσπαθώντας να ξεγελάσει το θύμα του και να κλέψει τις πληροφορίες που θέλει.
- Ένα άλλο είδος είναι η επανεκτέλεση μηνύματα (replay attack), κατά την οποία ο επιτεθειμένος λαμβάνει τα μηνύματα από τον αποστολέα και τα επαναπροωθεί αργότερα στον παραλήπτη δημιουργώντας προβλήματα στην επικοινωνία του με τον αποστολέα.
- Ακόμη ένα είδος είναι η τροποποίηση του μηνύματος με σκοπό την τελική αλλοίωση του νοήματος του κειμένου που αποστέλνεται.
- Άλλο ένα είδος είναι η επίθεση άρνησης εξυπηρέτησης (Denial Of Services), που συμβαίνει σε εξυπηρετητές, όπου ο επιτιθέμενος στέλνει πολλά πακέτα προς τον στόχο του με αποτέλεσμα να μην μπορεί να εξυπηρετήσει τους χρήστες, ή και πολλές φορές την κατάρρευση του ίδιου του συστήματος.
- Τέλος ένα είδος επιθέσεων είναι οι Ιοί (Virus), που στόχο έχουν την τροποποίηση αρχείων και πληροφοριών, ή την αποστολή πληροφοριών στους επιτιθέμενους.

## 1.2 Πως Αντιμετωπίζεται η Ασφάλεια στο Διαδίκτυο

Λόγω της δομής του, το διαδίκτυο αποτελεί το κύριο μέσο ανταλλαγής πληροφοριών. Καθώς πληθαίνουν οι χρήστες του διαδικτύου, πληθαίνουν και οι χρήστες που θα

προσπαθήσουν να αποσπάσουν πληροφορίες από τις συναλλαγές άλλων χρηστών στο διαδίκτυο. Για να εξασφαλιστεί η ασφάλεια στο διαδίκτυο θα πρέπει να τηρούνται τρείς βασικές απαιτήσεις. Σύμφωνα με το NIST (National Institute of Standards and Technology) οι τρεις αυτές αρχές, γνωστές ως CIA triad είναι θεμελιώδεις για την ασφάλεια στο διαδίκτυο και παρουσιάζονται στην εικόνα 2 (Stallings, W., 2010, σσ. 3-4).

- **Εμπιστευτικότητα (Confidentiality)**: είναι έννοια άρρηκτα συνδεδεμένη με την ιδιωτικότητα και ο σκοπός της είναι η μη αποκάλυψη των πληροφοριών σε μη εξουσιοδοτημένους χρήστες.
- **Ακεραιότητα (Integrity)**: προσφέρει προστασία των πληροφοριών από την τροποποίηση ή καταστροφή τους από μη εξουσιοδοτημένους χρήστες.
- **Διαθεσιμότητα (Availability)**: αναφέρεται στην άμεση, αδιάκοπη, χωρίς καθυστερήσεις, και αξιόπιστη πρόσβαση στις αιτούμενες εξουσιοδοτημένες υπηρεσίες και πληροφορίες (CCITT, 1991) (Shirey, 2000) (Stallings, W., 2010, σσ. 3-4) (Κατσίδου, Μ., 2007, σ. 16) (Μάντζιος, Ι., Μπαντίας, Σ., 2007, σσ. 8-9).




**Εικόνα 2.** Η τριάδα ασφαλείας στο Διαδίκτυο (Stallings, W., 2010).

Επιπλέον, υπάρχουν ακόμα απαιτήσεις, οι οποίες αν πληρούνται, θεωρείται ότι σου παρέχεται η μέγιστη ασφάλεια στις συναλλαγές σου στο διαδίκτυο. Αυτές είναι:

- **Αυθεντικοποίηση (Authentication)**: αναφέρεται στην διαδικασία κατά την οποία γίνεται ταυτοποίηση μιας οντότητας ότι όντως είναι αυτή που ισχυρίζεται ότι είναι.
- **Αποποίηση των Ευθυνών (Accountability)**: αναφέρεται στο γεγονός ότι κάθε οντότητα είναι υπεύθυνη για τις πράξεις της και την ασφάλειά της, και ως τέτοια θα πρέπει να δρα στο διαδίκτυο.

Ένα βασικό ζήτημα στην ασφάλεια του διαδικτύου είναι το τι επιλέγει κάποιος να προστατέψει. Δεν είναι δυνατόν να προστατευτούν όλες τις πληροφορίες, και ούτε είναι δυνατόν αυτό να πραγματοποιηθεί με τη μέγιστη ασφάλεια που μπορεί να υπάρξει. Για το λόγω αυτό γίνεται μια μελέτη για το πόσο καταστροφικές θα είναι οι συνέπειες από ενδεχόμενες απώλειες πληροφοριών και ποιο θα είναι το αντίκτυπό τους, και αναλόγως διαβαθμίζεται το μέγεθος του κινδύνου.

- Πρώτη διαβάθμιση είναι το **χαμηλό (Low)**, όπου η απώλεια των πληροφοριών αυτών μπορεί να εκθέσει σε περιορισμένο, και ενίοτε ελεγχόμενο κίνδυνο τον οργανισμό ή τους χρήστες.
- Δεύτερη διαβάθμιση είναι το **μέτριο (Moderate),** όπου η απώλεια δεδομένων, μπορεί να έχει σοβαρές επιπτώσεις στον οργανισμό και στους χρήστες.
- Τέλος η ανώτατη διαβάθμιση είναι το **μέγιστο (high)**, όπου πιθανότατη απώλεια των πληροφοριών μπορεί να έχει καταστροφικές συνέπειας, φτάνοντας μέχρι και το κλείσιμο του οργανισμού.

Η εφαρμογή των πιο πάνω απαιτήσεων και η προστασία των πληροφοριακών συστημάτων, παρουσιάζει δυσκολίες και είναι περίπλοκη. Όσο πιο προστατευμένο είναι το σύστημα μας, τόσο πιο δύσκολο, περίπλοκο και χρονοβόρο είναι για τους χρήστες να έχουν πρόσβαση και στις πληροφορίες για τις οποίες έχουν εξουσιοδότηση. Επίσης πιο περίπλοκη είναι και η επίλυση των οποιονδήποτε προβλημάτων εμφανιστούν στο σύστημα. Η εφαρμογή των μέτρων ασφαλείας θα πρέπει να γίνει κατά το σχεδιασμό του συστήματος, ειδάλλως το σύστημα θα είναι δύσκολο να υποστηρίξει το επιθυμητό επίπεδο ασφαλείας αργότερα. Τέλος, αν αναλογιστούμε, ότι η ασφάλεια υπολογιστικών συστημάτων, είναι μια “διαρκής μάχη”, ανάμεσα στους σχεδιαστές του συστήματος και τους επιτιθέμενους, μπορούμε να θεωρήσουμε ότι οι επιτιθέμενοι έχουν σοβαρό πλεονέκτημα έναντι των σχεδιαστών, αφού χρειάζεται να βρουν ένα κενό ασφαλείας για να δημιουργήσουν ζημία, αντιθέτως οι σχεδιαστές είναι υποχρεωμένοι να βρουν όλα τα κενά ασφαλείας πριν αυτά γίνουν γνωστά στους επιτιθέμενους (CCITT, 1991) (Shirey, 2000) (Stallings, W., 2010, σσ. 5-8) (Stallings, W., 2014, pp. 9-14).

## 1.3 Πως Επιτυγχάνεται η Ασφάλεια στο Διαδίκτυο

Η ασφάλεια των πληροφοριακών συστημάτων επιτυγχάνεται μέσω ορισμένων μηχανισμών που αναπτύχθηκαν με σκοπό την ικανοποίηση μιας ή περισσοτέρων

απαιτήσεων ασφαλείας, όπως έχουν περιγραφεί και πιο πάνω. Οι μηχανισμοί ασφαλείας παρουσιάζονται παρακάτω:

- **Κρυπτογράφηση (Encryption)**: Είναι η διαδικασία μετατροπής του απλού κειμένου σε μη-αναγνώσιμη μορφή, χρησιμοποιώντας ένα κλειδί και με τη βοήθεια κάποιου αλγορίθμου. Με την κρυπτογραφία επιτυγχάνουμε την εξασφάλιση της εμπιστευτικότητας.
- **Ψηφιακή Υπογραφή (Digital Signature)**: Αποτελεί τη διαδικασία πιστοποίησης των πληροφοριών που ανταλλάσσονται. Με την ψηφιακή υπογραφή επιτυγχάνουμε την ακεραιότητα, μιας και είμαστε σε θέση να γνωρίζουμε ότι τα δεδομένα που ανταλλάσσονται δεν έχουν υποστεί οποιαδήποτε τροποποίηση.
- **Έλεγχος Προσπέλασης (Access Control)**: Είναι ο έλεγχος των πληροφορίων που μπορεί να έχει πρόσβαση ο εκάστοτε χρήστης, μέσω ειδικών μηχανισμών προστασίας, που είναι συνήθως λίστες ελέγχου ή ετικέτες ασφαλείας, για τον καθορισμό των ορίων πρόσβασης σε κάθε χρήστη.
- **Ακεραιότητα Δεδομένων (Data Integrity)**: Είναι η εξασφάλιση της ακεραιότητας των δεδομένων, μέσω διαφόρων μηχανισμών ακεραιότητας.
- **Αυθεντικοποίηση (Authentication)**: Είναι η ο μηχανισμός ταυτοποίησης της οντότητας με την οποία γίνεται ανταλλαγή πληροφορίων, ότι όντως είναι η οντότητα που ισχυρίζεται ότι είναι, άρα και τα δεδομένα που ανταλλάσσονται είναι με τη συγκεκριμένη οντότητα. Με την αυθεντικοποίηση εξασφαλίζουμε την διαθεσιμότητα και την αυθεντικοποίηση.

Αυτοί είναι οι βασικοί μηχανισμοί, τους οποίους χρησιμοποιούμε για να προστατέψουμε την επικοινωνία των πληροφοριακών μας συστημάτων και κάποιοι απ' αυτούς θα αναλυθούν εκτενέστερα παρακάτω στα πλαίσια της Διατριβής μου (CCITT, 1991) (Stallings, W., 2010, σσ. 13-19) (Stallings, W., 2014, pp. 17-21) (Κατσίδου, Μ., 2007, σσ. 27-31).

# Κεφάλαιο 2

# Κρυπτογραφία

Η κρυπτογραφία είναι ο κλάδος των μαθηματικών που επιχειρεί να παρέχει ασφάλεια στην ανταλλαγή πληροφοριών καλύπτοντας τις προαναφερόμενες απαιτήσεις, δηλ. εμπιστευτικότητα, ακεραιότητα, διαθεσιμότητα, αυθεντικοποίηση και αποποίηση των ευθυνών, μέσω του σχεδιασμού κατάλληλων αλγορίθμων και πρωτοκόλλων. Είναι μια σύνθετη λέξη και προέρχεται από το συνδυασμό δυο λέξεων. Της λέξης κρυπτό που σημαίνει κρυφό, μυστικό και της λέξης Γράφος που σημαίνει γραφή. Δεν είναι μια καινούργια επιστήμη, αντιθέτως, έχει τις ρίζες πολύ παλαιά στην αρχαιότητα, όπου οι Αρχαίοι Αιγύπτιοι την χρησιμοποίησαν πρώτα και μετέπειτα οι Αρχαίοι Έλληνες, (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007, σ. 24).

Η κρυπτογραφία δεν πρέπει να συγχέεται με την κρυπτανάλυση. Η κρυπτανάλυση είναι κι αυτή επιστήμη που έχει ως στόχο τη ανάλυση των κρυπταλγορίθμων (και την εύρεση ευπαθειών που θα οδηγήσουν στην καταστρατήγηση ενός ή περισσοτέρων στόχων ασφάλειας) με τη γνώση κάποιων στοιχείων, όπως κομμάτια κρυπτογραφημένου κειμένου, ή κομμάτια του απλού κειμένου, ή το τον αλγόριθμο κρυπτογράφησης, ή και συνδυασμό των παραπάνω.

Και οι δυο παραπάνω περιοχές αναφέρονται συλλογικά ως κρυπτολογία. Οι βασικές έννοιες που χρησιμοποιούμε είναι οι εξής (CCITT, 1991) (SHIREY, 2000) (Stallings, W.,

2010, pp. 29-31) (Stallings, W., 2014, σσ. 29-32) (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007, p. 24):

- **Απλό Κείμενο (Plaintext)**: Το αρχικό μήνυμα το οποίο αποτελεί είσοδο για τον αλγόριθμο κρυπτογράφησης και υπόκεινται σε επεξεργασία ούτως ώστε να μην μπορεί να διαβαστεί έστω και αν βρεθεί σε λάθος χέρια.
- **Αλγόριθμος κρυπτογράφησης (Encryption algorithm)**: Είναι ο αλγόριθμος εκείνος κατά τον οποίο, δεδομένου ενός μυστικού κλειδιού, κάνει τις κατάλληλες πράξεις, μετασχηματίζοντας και μεταθέτοντας τα bits του μηνύματος, αναλόγως, μας δίδει στην έξοδο του το κρυπτογραφημένο μήνυμα.
- **Μυστικό κλειδί (Secret key)**: Αποτελεί τη δεύτερη είσοδο στον αλγόριθμο κρυπτογράφησης. Πρέπει να επιλεγεί πολύ προσεκτικά καθώς το μέγεθος του κλειδιού καθορίζει των αριθμό των πράξεων που θα τελεστούν από τον αλγόριθμο κρυπτογράφησης, καθώς και το πλήθος των μετασχηματισμών και μεταθέσεων.
- **Κρυπτογραφημένο μήνυμα (Ciphertext)**: Το αποτέλεσμα της εξόδου του αλγόριθμου κρυπτογράφησης, το οποίο εξαρτάται από το μυστικό κλειδί. Διαφορετικά μυστικά κλειδιά, παράγουν διαφορετικό κρυπτογραφημένο μήνυμα, δοθέντος ίδιου απλού κειμένου και αλγορίθμου κρυπτογράφησης. Το κρυπτογραφημένο μήνυμα αποτελεί είσοδο του αλγόριθμου αποκρυπτογράφησης.
- **Αλγόριθμος αποκρυπτογράφησης (Decryption Algorithm)**: Είναι ο αλγόριθμος που έχοντας ως είσοδο το κρυπτογραφημένο μήνυμα και το μυστικό κλειδί, παράγει με αντίστροφη διαδικασία από τον αλγόριθμο κρυπτογράφησης το αρχικό απλό κείμενο.

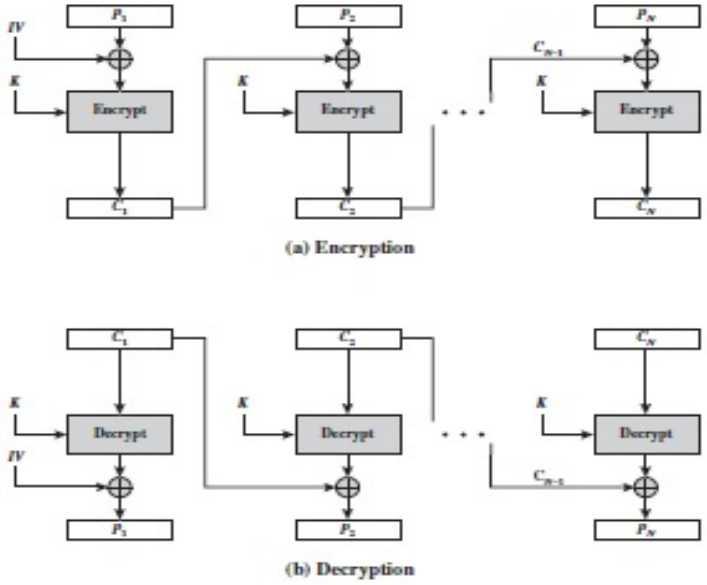
## 2.1 Τυχαίοι και Ψευδοτυχαίοι Αριθμοί

Οι τυχαίοι αριθμοί αποτελούν βασικό κομμάτι της κρυπτογραφίας χρησιμοποιούνται ευρέως από διάφορες εφαρμογές στο διαδίκτυο. Η επιτυχία της ύπαρξης τους βασίζεται στο γεγονός της τυχαιότητάς τους και της μη προβλεψιμότητας τους, διότι ο κάθε αριθμός που παράγεται είναι ανεξάρτητος από τον προηγούμενο του.

Πιο γνωστή και ευρέως χρησιμοποιούμενη γεννήτρια τέτοιων αριθμών είναι η γεννήτρια ψευδοτυχαίων αριθμών (PseudoRandom Number Generator – PRNG), η οποία χρησιμοποιεί μια σταθερή ποσότητα seed ως είσοδο, και την ανατροφοδότηση της εξόδου της για να παραγάγει τους ψευδοαριθμούς. Από την άλλη η ψευδοτυχαία συνάρτηση (PseudoRandom Function (PRF)), χρησιμοποιείται για την παραγωγή σταθερών ακολουθιών. Χρησιμοποιείται κυρίως από εφαρμογές του διαδικτύου, όπως SSL, TLS, και IPsec μεταξύ άλλων, για την παραγωγή προσωρινών κλειδιών συνόδου, καθώς και από το σύστημα Kerberos. Η γεννήτριες ψευδοτυχαίων αριθμών βρίσκουν εφαρμογές σε συναρτήσεις κατακερματισμού (hash functions), σε κώδικες αυθεντικοποίησης μηνυμάτων (Message Authentication Codes – MAC), στην ασύμμετρη κρυπτογραφία, π.χ. RSA key exchange, και στη συμμετρική κρυπτογραφία, π.χ. ρυθμός λειτουργίας CBC (Stallings, W., 2010, pp. 43-45) (Stallings, W., 2014, pp. 183-185).

## 2.2 Ρυθμός Λειτουργίας CBC (Cipher Block Chaining)

Στην περίπτωση που θέλουμε να κρυπτογραφήσουμε τμήμα δεδομένων, τα οποία είναι μεγαλύτερα των στάνταρντ τμήμα που χρησιμοποιούν οι συνηθισμένοι συμμετρικοί αλγόριθμοι κρυπτογράφησης (DES 64-bits, 3DES 64-bits, AES 128-bits), τότε μπορούμε να χρησιμοποιήσουμε τους ρυθμούς λειτουργίας (cipher block mode of operation). Τα βασικά τους χαρακτηριστικά είναι η ανθεκτικότητα, διάχυση σφαλμάτων, επανασυγχρονισμός και αποδοτικότητα. Υπάρχουν πέντε τύποι ρυθμών λειτουργίας ECB (Electronic CodeBook), CBC (Cipher Block Chaining), που είναι ο πιο γνωστός, πολυχρησιμοποιημένος και θα αναλυθεί εκτενέστερα στα πλαίσια της Διατριβής, CFB (Cipher FeedBack), OFB (Output FeedBack) και CTR (CounTeR).




**Εικόνα 3.** Σχεδιάγραμμα ρυθμού λειτουργίας CBC κατά την κρυπτογράφηση και την αποκρυπτογράφηση (Stallings, W., 2010).

Στο ρυθμό CBC η κρυπτογράφηση γίνεται παίρνοντας το απλό κείμενο μαζί με το προηγούμενο κρυπτογραφημένο τμήμα μέσω της πράξης XOR, όπως φαίνεται και στην εικόνα 3a. Επομένως, δυο ίδια τμήματα απλού κειμένου δεν μπορούν να οδηγήσουν στα ίδια τμήματα κρυπτογραφημένου κειμένου. Έτσι, για την αποκρυπτογράφηση, φαίνεται στην εικόνα 3b, πρέπει να πάρουμε τα τμήματα από την αρχή και με την σειρά, αλλιώς δε θα έχουμε τα επιθυμητά αποτελέσματα. Για να ξεκινήσει η κρυπτογράφηση χρειαζόμαστε ένα διάνυσμα αρχικοποίησης IV, το οποίο προκύπτει από τους τυχαίους ψευδοαριθμούς που αναφέραμε στο προηγούμενο υποκεφάλαιο. Χωρίς το διάνυσμα αυτό δεν είναι δυνατή η αποκρυπτογράφηση του τμήματος, και ως εκ τούτου πολλές φορές κρίνεται αναγκαία η κρυπτογράφηση του ίδιου του διανύσματος με κάποιον άλλο αλγόριθμο και μετά η ανταλλαγή του. Αν συμβεί λάθος σε ένα τμήμα τότε αυτό είναι αυτοδιορθούμενο και θα επιδιορθωθεί μετά από δυο τμήματα (Stallings, W., 2010, σσ. 50-52) (Stallings, W., 2014, pp. 183-185) (Κολοκοτρώνης Ν. , 2014, σσ. 41-42).

Οι εξισώσεις που διέπουν την κρυπτογράφηση και αποκρυπτογράφηση αντιστοίχως είναι:

$$C_i = E_K\left(P_i \oplus C_{i-1}\right)$$

(2.1)

$$P_i = D_K\left(C_i\right) \oplus C_{i-1}$$

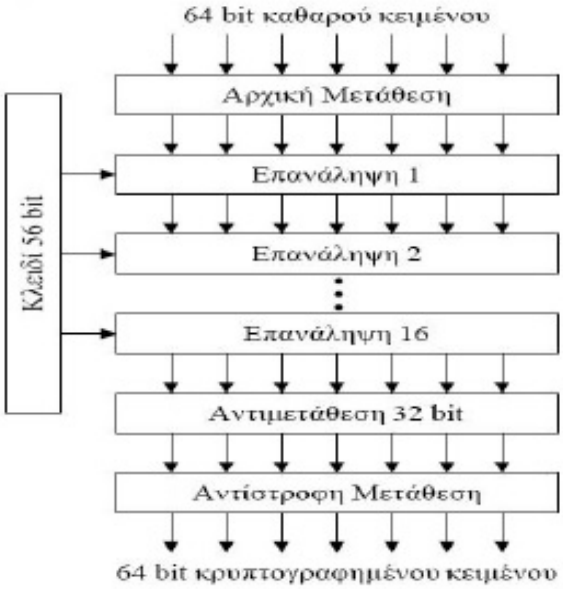
(2.2)

## 2.3 Συμμετρική Κρυπτογραφία

Στη συμμετρική κρυπτογραφία το κλειδί για την κρυπτογράφηση και αποκρυπτογράφηση είναι το ίδιο. Διέπεται από δυο βασικές προϋποθέσεις. Πρώτον ο αλγόριθμος κρυπτογράφησης, να είναι ένας ισχυρός και πολύπλοκος αλγόριθμος. (William, M., D., Raymont, G., K., 1999)Παρόλα αυτά δεν χρειάζεται να κρατηθεί μυστικός. Δεύτερη προϋπόθεση είναι η διατήρηση του κοινού κλειδιού, μυστικό. Το κλειδί πρέπει να το γνωρίζουν μόνο τα εξουσιοδοτημένοι χρήστες. Έτσι το κυριότερο πρόβλημα της συμμετρικής κρυπτογραφίας είναι η διανομή και διάδοση του κλειδιού στους εξουσιοδοτημένους χρήστες. Οι κυριότεροι αλγόριθμοι συμμετρικής κρυπτογραφίας είναι DES (Data Encryption Standard), 3DES (Triple Data Encryption Standard) και AES (Advance Encryption Standard).

### 2.3.1 DES (Data Encryption Standard)

Ένας από τους ευρέως χρησιμοποιημένους αλλά και πολυσυζητημένους αλγόριθμους συμμετρικής κρυπτογράφησης, ο Data Encryption Algorithm (DES), αναπτύχθηκε στα μέσα της δεκαετίας του 70 από το NIST. Οι προδιαγραφές του αλγορίθμου DES δημοσιεύτηκαν στην αναφορά FIPS PUB 46-3 (William, M., D., Raymont, G., K., 1999). Υπάρχουν υπόνοιες, χωρίς ωστόσο να έχουν αποδεχτεί επίσημα, ότι ο NIST (και συγκεκριμένα η NSA) επηρέασε το αλγόριθμο.



**Εικόνα 4. Αρχιτεκτονική αλγορίθμου DES** (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007).

Ο αλγόριθμος χρησιμοποιεί ένα κλειδί μεγέθους 64-bit (για την ακρίβεια 56-bit καθώς τα 8 bit χρησιμοποιούνται ως bit ισοτιμίας), και πραγματοποιεί 16 επαναλήψεις, όπως απεικονίζεται στην εικόνα 4. Σε κάθε επανάληψη χρησιμοποιείται ένα διαφορετικό υποκλειδί (που ονομάζεται κλειδί γύρου) μήκους 48-bit. Ο DES κρυπτογραφεί τμήμα των 64-bit, και μερικές φορές χρησιμοποιείται με το CBC, που αναφέρθηκε στο προηγούμενο υποκεφάλαιο, για να αυξήσει την ασφάλεια του, μιας και το κάθε κρυπτογραφημένο τμήμα θα εξαρτάται από το προηγούμενο. Οι αμφισβητήσεις που δέχτηκε, καθώς και το μικρό μέγεθος του μυστικού κλειδιού, οδήγησαν σε αναθεώρηση και ανασχεδιασμό του αλγορίθμου.

### 2.3.2 TDES ή 3DES (Triple Data Encryption Standard)

Ο κώδικας συμμετρικής κρυπτογράφησης TDES ή αλλιώς 3DES, χρησιμοποιεί τον αλγόριθμο κρυπτογράφησης DES, που αναφέρθηκε για πρώτη φορά στο προηγούμενο κεφάλαιο, αλλά χρησιμοποιώντας τρία κλειδιά, εξού και το όνομά του. Μια πρώτη αναφορά για τον 3DES, συναντάμε στο fips-pub 46-3 το 1999 (William, M., D., Raymont, G., K., 1999).

Όπως αναφέρθηκε ο 3DES χρησιμοποιεί τρία κλειδιά, και ακολουθεί τη διαδοχή κρυπτογράφηση-αποκρυπτογράφηση-κρυπτογράφηση. Οι σχέσεις που αποτυπώνουν την κρυπτογράφησης και την αποκρυπτογράφηση του αλγορίθμου φαίνονται στις εξισώσεις 2.3 και 2.4. Τα κλειδιά έχουν μήκος 56-bit, και επομένως το ωφέλιμο μήκος κλειδιού φτάνει τα 168-bit, πράγμα καθιστά τον κώδικα ισχυρό, μιας και είναι δύσκολη η εύρεση του κλειδιού απ' τον επιτιθέμενο. Αν τα κλειδιά $K_1 = K_3$, τότε το ωφέλιμο μήκος του κλειδιού μειώνεται στα 112-bit. Επίσης, αν $K_1 = K_2$, τότε ο 3DES = DES.

$$C = E_{K3}\left(D_{K2}\left(E_{K1}(P)\right)\right)$$

(2.3)

$$P = D_{K1}\left(E_{K2}\left(D_{K3}(C)\right)\right)$$

(2.4)

όπου:

C = κρυπτογραφημένο κείμενο

P = απλό κείμενο

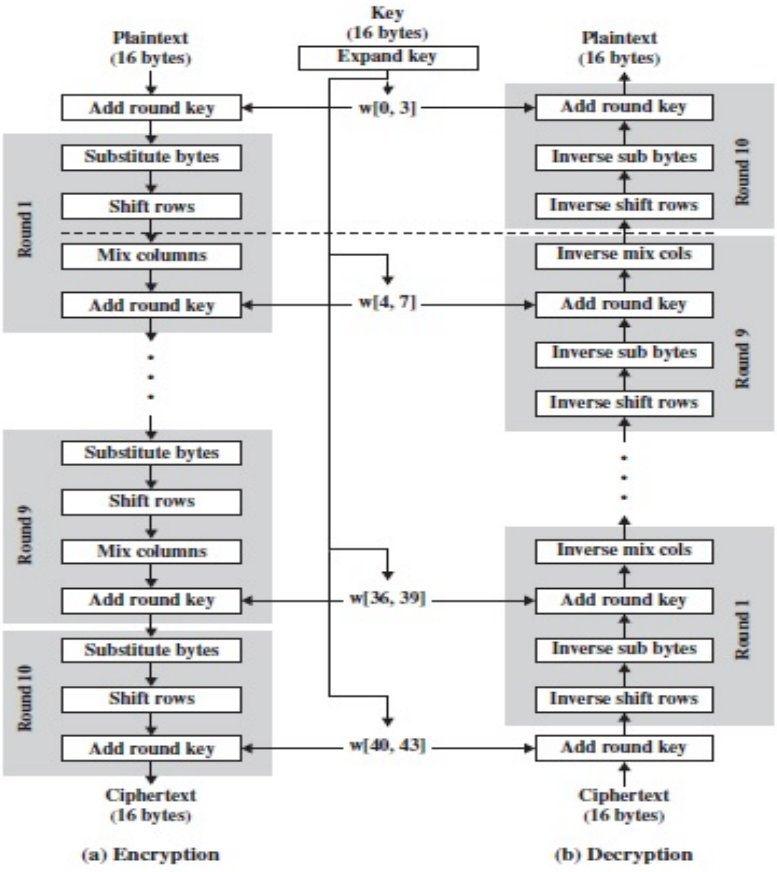
$E_K(X)$ = κρυπτογράφηση του X με το κλειδί K

$D_K(X)$ = αποκρυπτογράφηση του X με το κλειδί K.

Παρόλο που ο 3DES, είναι πολύ ισχυρός, λόγω της πολλαπλής χρήσης του αλγορίθμου DES (και συνεπώς έχει μεγαλύτερο κλειδί), και θεωρείται ότι θα επικρατούσε στο μέλλον, έχει ένα σοβαρότατο μειονέκτημα. Είναι πολύ αργός στην υλοποίησή του, αφού περιέχει τρεις περισσότερους γύρους από τον ήδη βραδύ DES. Επιπλέον μειονέκτημα, αποτελεί το γεγονός, ότι η κρυπτογράφηση γίνεται σε τμήμα δεδομένων, μέχρι τα 64-bit, που για λόγους ασφαλείας και αποδοτικότητας είναι επιθυμητό μεγαλύτερο μέγεθος (Stallings, W., 2010, σσ. 29-38) (William, M., D., Raymont, G., K., 1999) (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007, σσ. 29-38) (Γκολέμη, 2010, pp. 33-34) (Κατσίδου, Μ., 2007, σσ. 37-38) (Κολοκοτρώνης Ν. , 2014, σσ. 48-82).

### 2.3.3 AES (Advance Encryption Standard)

Βάση των παραπάνω αδυναμιών του DES και 3DES, ο NIST προκήρυξε διαγωνισμό πρότασης για τον AES. Ο καινούργιος αλγόριθμος θα έπρεπε να είναι το ίδιο αποδοτικός και ισχυρός, αλλά επιπλέον θα έπρεπε να έχει λιγότερες απαιτήσεις σε μνήμη και λιγότερες απαιτήσεις σε υλικό και λογισμικό για την υλοποίησή του. Τελικά νικητής του διαγωνισμού αναδείχθηκε η πρόταση των Βέλγων κρυπτογράφων Δρ. Joan Daemen και του Δρ. Vincent Rijmen και το τελικό πρότυπο δημοσιεύτηκε το 2001 στο fips pub 197 (Daemen, J., Rijmen, V., 2001).




**Εικόνα 5.** Κρυπτογράφηση και αποκρυπτογράφηση του αλγορίθμου AES (Stallings, W., 2010).

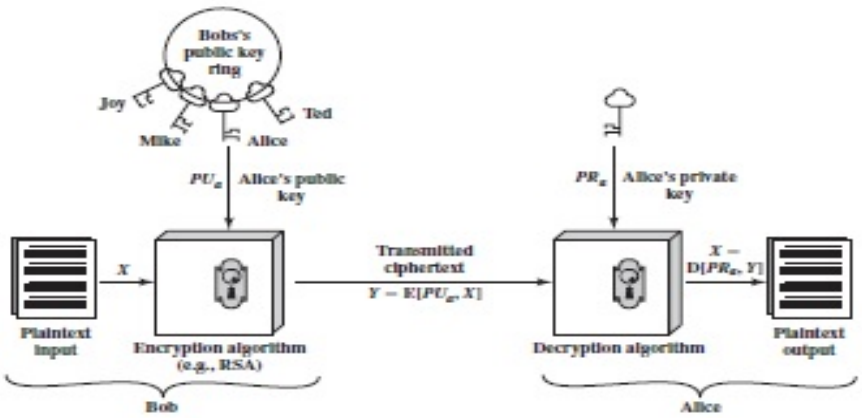
Ο AES κρυπτογραφεί τμήμα των 128-bit και έχει τη δυνατότητα επιλογής κλειδιού κρυπτογράφησης μεταξύ των 128-bit, 192-bit και 256-bit. Χρησιμοποιεί πίνακες, όπως φαίνεται στην εικόνα 5, από bytes, μεγέθους 4x4, για να κάνει κάποιες επαναλήψεις, των οποίων ο αριθμός τους εξαρτάται από το μέγεθος του κλειδιού. Το κλειδί που χρησιμοποιείται σε κάθε επανάληψη, προκύπτει από το αρχικό κλειδί. Τέλος σε κάθε επανάληψη, συμβαίνουν τα εξής, αντικατάσταση των bytes με τη χρήση S-box, ολίσθηση γραμμών, ανάμειξη δεδομένων σε κάθε στήλη του πίνακα, την πράξη XOR του πίνακα με το κλειδί (Daemen, J., Rijmen, V., 2001) (Stallings, W., 2010, σσ. 38-41) (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007, σσ. 38-39) (Κολοκοτρώνης Ν. , 2014, σσ. 83-91).

## 2.4 Ασύμμετρη Κρυπτογραφία

Το 1976 ο W. Diffie και M. Hellman προτείνουν για πρώτη φορά την έννοια της ασύμμετρης κρυπτογραφίας, δίνοντας μια νέα, διαφορετική άποψη για την έννοια της κρυπτογραφίας. Μια από τις διαφορές της ασύμμετρης, από τη συμμετρική κρυπτογραφία, είναι ότι ο αλγόριθμος που χρησιμοποιείται για την κρυπτογράφηση

χρησιμοποιεί μαθηματικές συναρτήσεις, εν αντιθέσει με τη συμμετρική όπου γίνονται πράξεις με bits. Ακόμα μια σημαντική διαφορά είναι ότι η ασύμμετρη κρυπτογραφία χρησιμοποιεί δυο κλειδιά, ενώ στη συμμετρική έχουμε ένα κοινό κλειδί.

Τα κλειδιά που χρησιμοποιούνται στην ασύμμετρη κρυπτογραφία ονομάζονται δημόσιο και ιδιωτικό, και δημιουργούνται πάντα ως ζεύγος. Το δημόσιο κλειδί είναι γνωστό σε όλους, το ιδιωτικό, όμως είναι μυστικό και το γνωρίζει μόνο ο κάτοχός του, ενώ συνήθως φυλάσσεται σε ασφαλές μέρος. Το ιδιωτικό κλειδί, αναφέρεται έτσι για να μην μπερδεύεται με το μυστικό κλειδί της συμμετρικής κρυπτογραφίας, δεν μπορεί σε καμιά περίπτωση να μπορεί να προκύψει από το δημόσιο (τουλάχιστον όχι σε πολυωνυμικό χρόνο), και γι' αυτό προστατεύεται ώστε να κρατηθεί μυστικό. Μπορεί να χρησιμοποιηθεί για την αποκρυπτογράφηση των δεδομένων. Αν για παράδειγμα έχουμε το Bob που θέλει να στείλει ένα μήνυμα στην Alice, αλλά θέλει, μόνο η Alice να μπορεί να αποκρυπτογραφήσει και να διαβάσει το μήνυμα. Τότε, ο Bob, όπως φαίνεται και στην εικόνα 4, κρυπτογραφεί το μήνυμα με  το δημόσιο κλειδί της Alice και στέλνει το μήνυμα στην Alice. Η Alice, και μόνο η Alice μπορεί να αποκρυπτογραφήσει το μήνυμα και να το διαβάσει, αφού είναι η μόνη που κατέχει το ιδιωτικό κλειδί.




**Εικόνα 6.** Κρυπτογράφηση μηνύματος μέσω ενός αλγορίθμου ασύμμετρης κρυπτογράφησης (Stallings, W., 2010).

Η ασύμμετρη κρυπτογραφία, παρέχει διάφορους τρόπους για την κάλυψη των βασικών αρχών της επικοινωνίας με τη χρήση, πότε του ιδιωτικού, και πότε του δημόσιου κλειδιού.

- **Κρυπτογράφηση/Αποκρυπτογράφηση (Encryption/Decryption):** Ο αποστολέας κρυπτογραφεί τα δεδομένα που θα αποστείλει με το δημόσιο κλειδί του παραλήπτη, και ο παραλήπτης τα αποκρυπτογραφεί με το ιδιωτικό του κλειδί.
- **Ψηφιακή Υπογραφή (Digital Signature):** Ο αποστολέας κρυπτογραφεί τα δεδομένα (δηλ. τη σύνοψη της προς μετάδοση πληροφορίας) με το ιδιωτικό του κλειδί, με τη βοήθεια μιας συνάρτησης κατακερματισμού, ή αλλιώς σύνοψης (hash function), και εν συνεχεία στέλνει τα δεδομένα στον παραλήπτη, ο οποίος αποκρυπτογραφεί τα δεδομένα με το δημόσιο του κλειδί, και πιστοποιεί τον αποστολέα.
- **Ανταλλαγή κλειδιών (Key Exchange):** Χρησιμοποιείται για την ανταλλαγή κλειδιού συνόδου (session key). Η διαδικασία αυτή ποικίλει και διαφέρει, και μπορεί να χρησιμοποιεί το ένα ή και τα δυο ιδιωτικά κλειδιά των εμπλεκόμενων μερών.

Κάποιοι αλγόριθμοι, μπορούν να χρησιμοποιηθούν και για τις τρεις, παραπάνω εφαρμογές, και κάποιοι άλλοι μόνο για κάποιες. Παρακάτω θα αναλυθούν οι πιο γνωστοί D-H (Diffie-Hellman) και RSA.

### 2.4.1 Αλγόριθμος RSA

Ο αλγόριθμος RSA, πήρε το όνομά του από τους R. Rivest, A. Shamir και L. Adleman, αναπτύχθηκε το 1977 στο MIT και δημοσιεύθηκε το 1978. Ο αλγόριθμος διέπεται από τις εξισώσεις (2.5), (2.6), για την κρυπτογράφηση και αποκρυπτογράφηση, αντίστοιχα. Ο ακέραιος *n*, που χρησιμοποιείται στην εξίσωση και είναι γνωστός, παράγεται από το γινόμενο δυο πρώτων αριθμών, διαφορετικών μεταξύ τους. Ο αλγόριθμος θεωρείται αρκετά ισχυρός, μιας και για να υπολογίσει κάποιος τη μυστική ποσότητα *d*, η οποία είναι ποσότητα που χρησιμοποιείται για τον υπολογισμό του ιδιωτικού κλειδιού, θα πρέπει να γνωρίζει τους δυο πρώτους αριθμούς (*p* και *q*), των οποίων το γινόμενό τους να μας δίδει το *n*. Το πρόβλημα αυτό στα μαθηματικά είναι γνωστό και ως πρόβλημα της παραγοντοποίησης μεγάλων ακεραίων (Stallings, W., 2010, pp. 79-85) (Stallings, W., 2014, pp. 254-275).

$$C = M^e \bmod (n)$$

(2.5)

$$M = C^d \bmod(n) = \left(M^e\right)^d \bmod(n) = M^{ed} \bmod(n)$$

(2.6)

όπου,

M = απλό κείμενο.

C = κρυπτογραφημένο κείμενο.

n = ακέραιος αριθμός, υπολογισμένος από το γινόμενο δυο πρώτων αριθμών.

e = ποσότητα που είναι μέρος του δημόσιου κλειδιού.

d = ποσότητα που είναι μέρος του ιδιωτικού κλειδιού.

Οι ποσότητες $n$ και $e$ είναι γνωστές μιας και χρησιμοποιούνται για να υπολογιστεί το δημόσιο κλειδί KU = {e, n}, ενώ η ποσότητα d, που χρησιμοποιείται για τον υπολογισμό του ιδιωτικού κλειδιού KR = {d, p, q}. Για να θεωρείται ισχυρός ο αλγόριθμος, θα πρέπει να είναι αδύνατο να υπολογιστεί η ποσότητα d δοθέντος των ποσοτήτων e και n. Για να συμβεί αυτό, οι πρώτοι αριθμοί πρέπει να είναι αρκετά μεγάλοι, και να έχουν μεγάλη διαφορά μεταξύ τους, ειδάλλως το $p \approx \sqrt{n}$, και το d μπορεί εύκολα να υπολογιστεί με ένα εξαντλητικό έλεγχο.

Για τον υπολογισμό των ποσοτήτων του αλγορίθμου RSA, θα πρέπει να πραγματοποιηθούν με τη σειρά οι παρακάτω πράξεις. Όπως αναφέρθηκε και παραπάνω το n υπολογίζεται από το γινόμενο των αριθμών p και q.

$$n = p \cdot q$$

(2.7)

Στη συνέχεια χρησιμοποιούμε τη συνάρτηση του Euler που φαίνεται στη σχέση (2.8), και επιλέγουμε στο e να είναι μεταξύ 1 και φ(n) στην αμέσως επόμενη σχέση.

$$\varphi(n) = (p-1)(q-1)$$

(2.8)

$$\gcd\left(\varphi(n), e\right) = 1; 1 < e < \varphi(n)$$

(2.9)

Αφού επιλέξουμε το e ούτως ώστε να είναι πρώτος αριθμός ως προς το φ(n), ακολούθως υπολογίζουμε το d από τη παρακάτω σχέση.

$$d = e^{-1} \bmod\left(\varphi(n)\right)$$

(2.10)

Επομένως μπορούμε να παράγουμε τα κλειδιά και να κρυπτογραφήσουμε και αποκρυπτογραφήσουμε δεδομένα. Ένα παράδειγμα, επιλέγουμε p = 47, q = 59 και m = 31 (Kaliski, B., Staddon, J.,, 1998) (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007) (Κατσίδου, Μ., 2007) (Κολοκοτρώνης Ν. , 2014).

$$n = p \cdot q = 47 \cdot 59 = 2773$$

$$\varphi(n) = (p-1)(q-1) = 2668$$

$$\gcd(\varphi(n), e) = 1; 1 < e < \varphi(n) = 17$$

$$d = e^{-1} \bmod (\varphi(n)) = 17^{-1} \bmod (2668) = 157$$

$$C = M^{e} \bmod (n) = 31^{17} \bmod (2773) = 587$$

$$M = C^{d} \bmod (n) = 587^{157} \bmod (2773) = 31$$

### 2.4.2 Αλγόριθμος ανταλλαγής κλειδιού Diffie-Hellman

Ένας από τους πρώτους αλγορίθμους που αναπτύχθηκαν βάση του ασύμμετρου συστήματος κρυπτογραφίας είναι ο αλγόριθμος ανταλλαγής κλειδιών Diffie-Hellman ο οποίος αναπτύχθηκε από τους W. Diffie και M. Hellman. Ο αλγόριθμος δημιουργήθηκε με σκοπό την ασφαλή ανταλλαγή του κλειδιού συνόδου που θα χρησιμοποιηθεί αργότερα για την κρυπτογράφηση της επικοινωνίας των χρηστών.

Για τον υπολογισμό του αλγορίθμου, παίρνουμε ένα πρώτο αριθμό p αρκετά μεγάλο και ακολούθως υπολογίζουμε τη πρωταρχική ρίζα του α. Οι ποσότητες αυτές είναι γνωστές σε όλους, μιας και ανταλλάσσονται μεταξύ του Α και του Β. Ακολούθως ο χρήστης Α επιλέγει μια αυθαίρετη ποσότητα ακεραίου $X_A$, με το μόνο περιορισμό $X_A < p$, και υπολογίζει βάση της σχέσης (2.11) την ποσότητα $Y_A$, την οποία ανταλλάσσει με το Β, επομένως είναι και γνωστή σε όλους. Από την άλλη ο Β επιλέγει μια αυθαίρετο ακέραιο $X_B$, όπου $X_B < p$, και υπολογίζει το $Y_B$ μέσω της (2.12), το οποίο ανταλλάσσει με τον Α και είναι κι αυτό γνωστό.

$$Y_A = \alpha^{X_A} \bmod (p)$$

(2.11)

$$Y_B = \alpha^{X_B} \bmod (p)$$

(2.12)

Τέλος, ο κάθε χρήστης υπολογίζει το κλειδί Κ, βάση των δικών του γνωστών ποσοτήτων.

$$
\begin{aligned}
\mathrm{K} &= (\mathrm{Y_B})^{X_A} \bmod(p) \\
&= \left(\alpha^{X_B} \bmod(p)\right)^{X_A} \bmod(p) \\
&= \left(\alpha^{X_B}\right)^{X_A} \bmod(p) = a^{X_A X_B} \bmod(p) \\
&= \left(\alpha^{X_A}\right)^{X_B} \bmod(p) \\
&= \left(\alpha^{X_A} \bmod(p)\right)^{X_B} \bmod(p) \\
&= \left(\mathrm{Y}_A\right)^{X_B} \bmod(p)
\end{aligned}
$$

(2.13)

Όπως φαίνεται και πιο πάνω, ο κάθε χρήστης μπορεί να υπολογίσει το κλειδί, βάση της μυστικής ποσότητας που ο ίδιος αυθαίρετα επέλεξε με μια πολύ εύκολη πράξη υπολοίπου. Αντιθέτως ο επιτιθέμενος, ο οποίος γνωρίζει τις ποσότητες α, p, $Y_A$, $Y_B$, είναι δύσκολο να υπολογίσει το κλειδί γιατί δεν γνωρίζει κάποια από τις μυστικές ποσότητες. Επομένως, αν υποθέσουμε ότι επιτίθεται στον B, τότε θα πρέπει να λογαριθμίσει για να τον υπολογισμό βάση της σχέση (2.14), πράξη σαφέστατα δυσκολότερη από την προηγούμενη.

$$\mathrm{X_B} = d\log_{\alpha,p}\left(\mathrm{Y}_B\right)$$

(2.14)

Έτσι, το πρόβλημα του Diffie-Hellman είναι γνωστό και ως πρόβλημα διακριτού λογαρίθμου, πράγμα που αυξάνει την ασφάλεια του.

Ως παράδειγμα εκτέλεσης του αλγορίθμου, θεωρούμε τις τιμές p=353 και την πρωταρχική ρίζα α=3. Αν ο A επιλέξει ένα αριθμό $X_A$=97 και ο B ένα αριθμό $X_B$=233 τότε:

$$\mathrm{A} \Rightarrow \mathrm{Y_A} = \alpha^{\mathrm{X_A}} \bmod(p) = 3^{97} \bmod(353) = 40$$

$$B \Rightarrow \mathrm{Y}_B = \alpha^{\mathrm{X}_B} \bmod(p) = 3^{233} \bmod(353) = 248$$

Αφού ανταλλάξουν τις ποσότητες $Y_A$ και $Y_B$, τότε:

$$\mathrm{A} \Rightarrow \mathrm{K} = (\mathrm{Y_B})^{X_A} \bmod(p) = 248^{97} \bmod 353 = 160$$

$$B \Rightarrow \mathrm{K} = \left(\mathrm{Y_A}\right)^{X_B} \bmod(p) = 40^{233} \bmod 353 = 160$$

Παρατηρούμε ότι το κλειδί που υπολόγισαν είναι το ίδιο και στις δυο περιπτώσεις.

Ο αλγόριθμος έχει ως μειονέκτημα το γεγονός ότι η άλλη πλευρά δε μπορεί να αυθεντικοποιηθεί και ο οποιοσδήποτε μπορεί να παριστάνει τον οποιονδήποτε υποκλέπτοντας πληροφορίες. Στο παράδειγμα του Bob με την Alice, ας υποθέσουμε ότι ο Darth ζηλεύει τον Bob και θέλει να ξέρει τι μηνύματα ανταλλάσσει με την Alice. Μετά την ανταλλαγή των ποσοτήτων α και p η Alice υπολογίζει και στέλνει την ποσότητα $Y_A$

στο Bob. Ο Darth όμως, σταματά το μήνυμα, επιλέγει δυο ποσότητες $X_{D1}$ και $X_{D2}$, και υπολογίζει και στέλνει το $Y_{D1}$ στο Bob. Ακολούθως ο Bob υπολογίζει και στέλνει στην Alice την ποσότητα $Y_B$. Ο Darth τη σταματά και υπολογίζει και στέλνει την ποσότητα $Y_{D2}$ στην Alice. Τώρα ο Darth και ο Bob υπολογίζουν το κλειδί K1 και η Alice και ο Darth το κλειδί K2. Ο Darth μπορεί να περιοριστεί απλώς στο να παρακολουθεί τη συνομιλία του Bob με την Alice, ή μπορεί ακόμη να επεμβαίνει παραποιώντας ή σταματώντας τα μηνύματα που ανταλλάσσονται. Εν τω μεταξύ η Alice και ο Bob θα πιστεύουν ότι θα συνομιλούν κάτω από ένα μυστικό ιδιωτικό κανάλι που θέσπισαν οι δυο τους και τα μηνύματά τους ότι δεν τροποποιούνται (Adrian, D., Bhargavan, K., Durumeric, Z., Caudry, P., Green, M., Halderman, A., J., Heninger, N., Springall, D., Thome, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanela-Beguelin, S., Zimmerman, P., 2015, pp. 2-3) (Rescorla, E., 1999) (Stallings, W., 2010, pp. 85-90) (Stallings, W., 2014, pp. 287-292) (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007, pp. 42-43) (Γκολέμη, 2010, σσ. 41-42).

# 2.5 Αυθεντικοποίηση

Για προστασία τόσο από τις παθητικές επιθέσεις (passive attacks), που αποτελούν παραβίαση της εμπιστευτικότητας των πληροφοριών από τους ωτακουστές, όσο και από τις ενεργητικές επιθέσεις (active attacks), που στοχεύουν στην τροποποίηση των πληροφοριών, χρειάζεται αυθεντικοποίηση. Η αυθεντικοποίηση δίνει τη δυνατότητα της εξασφάλισης της ακεραιότητας, δηλ. της μη τροποποίησης των μηνυμάτων, και επιπλέον την αυθεντικοποίηση-πιστοποίηση της πηγής. Ειδικότερα, αν το μήνυμα περιέχει και χρονοσήμανση, τότε σημαίνει ότι το μήνυμα δεν καθυστερεί περαιτέρω του φυσιολογικού στη μετάδοσή του. Δυο τρόποι με τους οποίους επιτυγχάνεται η αυθεντικότητα, είναι με τη χρήση κώδικα αυθεντικοποίησης μηνυμάτων (Message Authentication Code – MAC) και τις ψηφιακές υπογραφές (Digital Signatures).

## 2.5.1 Κώδικας Αυθεντικοποίησης Μηνυμάτων (MAC)

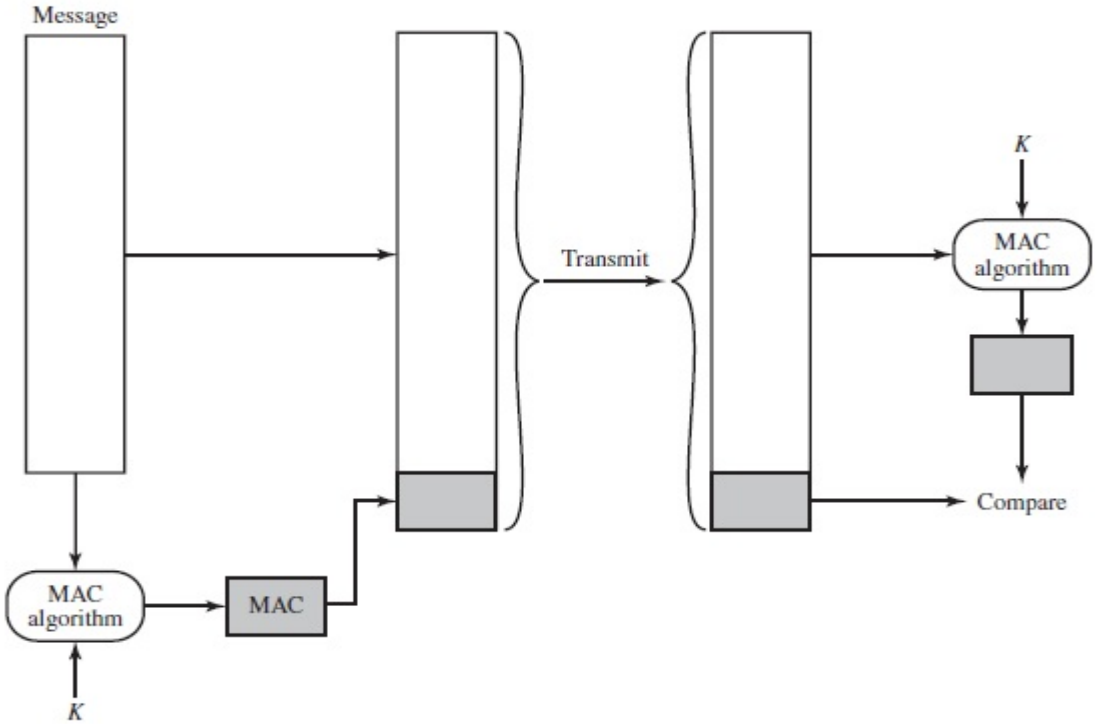
Είναι μια συνήθης και συχνή μέθοδος αυθεντικοποίησης, η οποία καλείται κώδικας αυθεντικοποίησης μηνυμάτων (Message Authentication Code – MAC). Για την εφαρμογή της μεθόδου αυτής είναι αναγκαία η γνώση από τις εμπλεκόμενες μεριές ενός μυστικού κλειδιού $K_{AB}$. Όταν η Alice θέλει να στείλει ένα μήνυμα στο Bob, υπολογίζει το MAC,

βάση μιας συνάρτησης που δέχεται ως παραμέτρους το κλειδί $K_{AB}$ και το μήνυμα M, σχέση (2.15).

$$MAC_M = f\left(K_{AB}, M\right)$$

(2.15)

Το MAC επισυνάπτεται στο μήνυμα και αποστέλνεται στο Bob, και εν συνεχεία ο Bob υπολογίζει και αυτός ένα MAC βάση του κλειδιού που κατέχει, όπως φαίνεται και στην εικόνα 7. Το MAC συγκρίνεται με αυτό του παραλήπτη, τότε τακτοποιείται ότι αυτός που έστειλε το μήνυμα είναι η Alice, μιας και κανένας άλλος δε γνωρίζει το μυστικό κλειδί $K_{AB}$. Προτιμότερος αλγόριθμος κρυπτογράφησης για το MAC ήταν ο DES, αλλά οποιοσδήποτε συμμετρικός αλγόριθμος μπορεί να χρησιμοποιηθεί (Stallings, W., 2010, pp. 73-79) (Stallings, W., 2014, pp. 357-367).




**Εικόνα 7.** Αυθεντικοποίηση μηνύματος με τη χρήση MAC (Stallings, W., 2010).

## 2.5.2 Ψηφιακή Υπογραφή (Digital Signature)

Η ψηφιακή υπογραφή χρησιμοποιείται για να παρέχει αυθεντικοποίηση του χρήστη. Χρησιμοποιεί μια συνάρτηση κατακερματισμού (hash function) και το ιδιωτικό κλειδί. Με τη ψηφιακή υπογραφή, επιτυγχάνουμε ακεραιότητα, επαλήθευση χρήστη και ταυτοποίηση του υπολογιστή. Για παράδειγμα η Alice θέλει να ταυτοποιηθεί από τον

Bob. Κατ' αρχάς παράγει ένα μήνυμα. Στη συνέχεια κατακερματίζει το μήνυμα με την βοήθεια μιας συνάρτησης κατακερματισμού. Μετά κρυπτογραφεί το κατακερματισμένο μήνυμα με το ιδιωτικό της κλειδί. Στην περίπτωση αυτή δεν μας ενδιαφέρει κάποιος να μην μπορεί να διαβάσει τα δεδομένα, παρά η απόδειξη ότι αυτός που στέλνει τα δεδομένα είναι η Alice, μιας και η Alice και μόνον αυτή κατέχει το ιδιωτικό κλειδί της Alice. Το κρυπτογραφημένο μήνυμα μαζί με το πρωτότυπο, στέλνονται στο Bob, ο οποίος αποκρυπτογραφεί κρυπτογραφημένο, με το δημόσιο κλειδί της Alice. Στη συνέχεια και με την ίδια συνάρτηση κατακερματισμού, ανακατασκευάζει το μήνυμα και το ελέγχει αν είναι ίδιο με το πρωτότυπο. Αν αυτό ισχύει, τότε η Alice είναι αυτή που του στέλνει τα μηνύματα.

Οι επιτυχία της ψηφιακής υπογραφής οφείλεται στη μυστικότητα του ιδιωτικού κλειδιού και στην συνάρτηση κατακερματισμού. Η συνάρτηση κατακερματισμού τεμαχίζει το μήνυμα σε μικρότερα κομμάτια σε μια μορφή που δείχνει τυχαία. Τα κομμάτια αυτά ονομάζονται hashes και έχουν σταθερό μήκος. Με το κατακερματισμό η πληροφορία του μηνύματος χάνεται καθώς δεν υπάρχει μέθοδος αποκωδικοποίησης ενός hash (δηλ. από το hash να μπορούμε να βρούμε το μήνυμα που δόθηκε στην είσοδο της συνάρτησης κατακερματισμού). Η σύνοψη του μηνύματος, λειτουργεί σαν ψηφιακό αποτύπωμα, μιας και μια μικρή αλλαγή στο μήνυμα προϋποθέτει υπολογισμό ξανά της σύνοψης για να μην προκύψει λάθος κατά την αποκρυπτογράφησή του και σύγκρισή του με το πρωτότυπο. Η πιο γνωστή συνάρτηση κατακερματισμού είναι η Secure Hash Algorithm (SHA-1), που σχεδιάστηκε από το NIST και παράγει hashes 160 bit (Stallings, W., 2010, p. 90) (Stallings, W., 2014, pp. 394-398) (Κατσίδου, Μ., 2007, pp. 52-53) (Κολοκοτρώνης Ν. , 2014, pp. 19-25) (Μάντζιος, Ι., Μπαντίας, Σ., 2007, σ. 14). Η εν λόγω συνάρτηση δε θεωρείται ασφαλής σήμερα. Έχουν όμως προ τυποποιηθεί άλλες συναρτήσεις στη συγκεκριμένη οικογένεια, π.χ. SHA-256, SHA-512, κ.α., οι οποίες θεωρούνται εξαιρετικά ασφαλείς.

## 2.6 Ψηφιακά Πιστοποιητικά (Digital Certificates)

Για την αποτελεσματική λειτουργία της ασύμμετρης κρυπτογραφίας χρειάζεται να γίνει διανομή του δημόσιου κλειδιού. Υπάρχουν διάφοροι μηχανισμοί για τη διάδοση των δημόσιων κλειδιών προς όλους τους ενδιαφερόμενους. Το κύριο ζήτημα είναι ότι ο Darth, που είναι ένας κακόβουλος χρήστης και θέλει να υποκλέψει τις πληροφορίες που

στέλνει η Alice στο Bob, μπορεί να στείλει ένα μη υπογεγραμμένο πιστοποιητικό και να παρουσιάζεται ως Bob, στέλνοντας το δημόσιο του κλειδί στην Alice. Η Alice, χωρίς να το γνωρίζει, θα στέλνει τα κρυπτογραφημένα μηνύματα στον Darth, ο οποίος θα τα αποκρυπτογραφεί με το ιδιωτικό του κλειδί. Αυτό οφείλεται στην αδυναμία της διασφάλισης της ακεραιότητας και αυθεντικοποίησης του αποστολέα.

Λύση στο παραπάνω πρόβλημα έρχονται να δώσουν τα υπογεγραμμένα πιστοποιητικά από μια Έμπιστη Τρίτη Οντότητα (Trusted Third Party – TTP), η οποία συνήθως καλείται και Πάροχος Υπηρεσιών Πιστοποίησης (Certification Service Provider – CSP). Ο όρος Αρχή Πιστοποίησης (Certification Authority – CA) χρησιμοποιείται πιο συχνά αν και υπονοεί ότι οι προσφερόμενες προς το χρήστη υπηρεσίες περιορίζονται σε αυτές που αφορούν τη διαχείριση ψηφιακών πιστοποιητικών. Οι Αρχές Πιστοποίησης μπορούν (και πρέπει) να συνεργάζονται ώστε να προσφέρουν ολοκληρωμένες υπηρεσίες προς τους χρήστες. Μια υποδομή δημοσίων κλειδιών (ΥΔΚ) περιλαμβάνει πλήθος Αρχών Πιστοποίησης οι οποίες οργανώνονται με διάφορους τρόπους. Ένα απλό (αλλά όχι σύνηθες) μοντέλο οργάνωσης είναι το ιεραρχικό μοντέλο, και το σύνθετο ιεραρχικό μοντέλο ως γενίκευση, όπου οι κύριες Αρχές Πιστοποίησης (root CAs), που βρίσκονται στην κορυφή της ιεραρχίας, δίνουν εξουσιοδοτήσεις σε άλλες Αρχές Πιστοποίησης για να μπορούν να δίνουν εξουσιοδοτήσεις εκ νέου, ή να εκδίδουν πιστοποιητικά. Κάθε Αρχή Πιστοποίησης, εκτός από την κύρια, βρίσκεται κάτω από μια ή περισσότερες Αρχές Πιστοποίησης.

Ένας χρήσης Α που θέλει να διαμοιραστεί το δημόσιο του κλειδί με άλλους χρήστες, στέλνει το πιστοποιητικό του, μαζί με κάποια λοιπά στοιχεία που βεβαιώνουν την ταυτότητα του, σε μια γνωστή εταιρία που είναι Αρχή Πιστοποίησης. Η εταιρία υπογράφει και πιστοποιεί το πιστοποιητικό. Όταν το στείλει σε κάποιον άλλο χρήστη Β, θα εκλάβει το πιστοποιητικό χωρίς κανένα πρόβλημα, μιας και η εταιρία είναι έμπιστη αρχή αφού υπόκεινται στην κύρια Αρχή Πιστοποίησης, και τα δημόσια κλειδιά της είναι γνωστά από όλους τους χρήστες. Ο χρήστης Β θα χρησιμοποιήσει το δημόσιο κλειδί του χρήστη Α, για να αποκρυπτογραφήσει την υπογραφή και να ταυτοποίηση το χρήστη Α.

### 2.6.1 Το Πιστοποιητικό X.509

Το πιο διαδεδομένο πρότυπο για τη σύνταξη ενός πιστοποιητικού είναι το X.509. Το X.509 δημοσιοποιήθηκε για πρώτη φορά το 1988 και αποτελεί σύσταση της Διεθνής Εταιρίας Τηλεπικοινωνιών (ITU). Είναι μέρος του ευρύτερου καταλόγου παροχής υπηρεσιών X.500.

Το X.509 είναι ένα πολύ σημαντικό πρότυπο, γιατί η δομή και αυθεντικοποίηση , που ορίζονται στο πρωτόκολλο αυτό, μπορεί να χρησιμοποιηθούν σε μια πληθώρα υπηρεσιών. Το X.509 μπορεί να χρησιμοποιηθεί στο S/MIME, IPSec, TLS/SSL.

Κατά την έκδοση ενός πιστοποιητικού βάση του προτύπου X.509, συμπληρώνονται κάποια στοιχεία, που παρατίθενται στον πίνακα 1, και είναι ορατά στο χρήστη (Stallings, W., 2010, σσ. 116-124) (Stallings, W., 2014, σσ. 435-443) (Κατσίδου, Μ., 2007, σσ. 56-59).

| Στοιχείο | Χρήση |
|---|---|
| Version | Είναι ένας αριθμός, από το 1 μέχρι το 3, που καθορίζει πόσα στοιχεία συμπεριλαμβάνονται και συμπληρώνονται. Το 1 είναι η προκαθορισμένη έκδοση και περιλαμβάνει τα λιγότερα στοιχεία, ενώ το 3 είναι η πιο πλήρης έκδοση. |
| Serial number | Ένας μοναδικός ακέραιος αριθμός που εκδίδεται από την Αρχή Πιστοποίησης και χρησιμοποιείται για την αναγνώριση του πιστοποιητικού. |
| Signature algorithm identifier | Αναφέρεται στον αλγόριθμο που χρησιμοποιείται για την υπογραφή του πιστοποιητικού μαζί με όποια άλλη πληροφορία έχει σχέση με αυτό. |
| Issuer name | Το όνομα της Αρχής Πιστοποίησης που χρησιμοποιήθηκε. |
| Period of validity | Αναφέρεται στην ημερομηνία ενεργοποίησης και ημερομηνία λήξης του πιστοποιητικού. |
| Subject name | Το όνομα της οντότητας που κατέχει το ιδιωτικό κλειδί. |
| Subject's public-key information | Το δημόσιο κλειδί της οντότητας. |
| Issuer unique identifier | Ο αριθμός αυτός χρησιμοποιείται σε συνδυασμό με το όνομα της Αρχής Πιστοποίησης για να ενισχύσει την αναγνώριση αυτής. |
| Subject unique identifier | Ο αριθμός αυτός χρησιμοποιείται για να προσδώσει μοναδικότητα στο πιστοποιητικό, αν η οντότητα κατέχει κι άλλο πιστοποιητικό |
| Extensions | Εδώ μπορούν να προστεθούν περισσότερα πεδία. Το πεδίο αυτό |

| | |
|---|---|
| | υπάρχει μόνο στην έκδοση 3. |
| Signature | Η Ψηφιακή υπογραφή με το ιδιωτικό κλειδί στις παραπάνω πληροφορίες. |

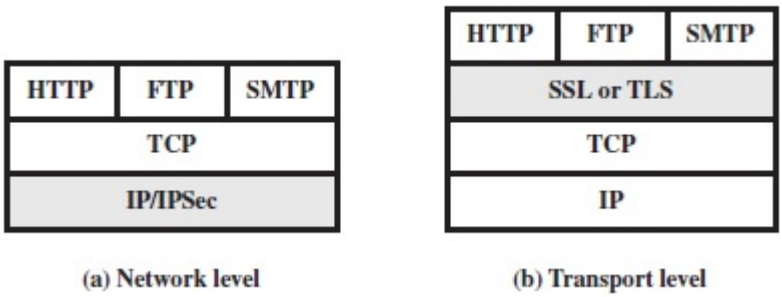
**Πίνακας 1.** Τα στοιχεία ενός πιστοποιητικού βάση του προτύπου X.509.

# Κεφάλαιο 3

# Πρωτόκολλα Ασφαλείας

Η ανάπτυξη του διαδικτύου, με την τεράστια ανταλλαγή πληροφοριών, το ηλεκτρονικό εμπόριο, την όλο και αυξανόμενη εισροή νέων εταιριών και χρηστών για την εκμετάλλευση των πληροφοριών και την ανταλλαγή καινούργιων, μόνο θετικά μπορεί να αποτιμηθεί. Παρόλα αυτά, η ασφάλεια και η ιδιωτικότητα, είναι δυο σημαντικά ζητήματα που απασχολούσαν πάντα την επιστημονική κοινότητα και τους χρήστες. Η εξασφάλιση των τριών αρχών ασφαλείας Ακεραιότητα, Εμπιστευτικότητα και Διαθεσιμότητα, είναι απαραίτητη (βλ. Κεφ. 1). Αυτό επιτυγχάνεται με την κρυπτογραφία. Η υλοποίηση της γίνεται μέσω των πρωτοκόλλων ασφαλείας που καθορίζουν που, πότε και τι θα εφαρμοστεί.

Η ασφάλεια στο διαδίκτυο, εφαρμόζεται στο TCP επίπεδο, τέταρτο επίπεδο του TCP/IP πρωτοκόλλου επικοινωνίας, με το πρωτόκολλο TLS (Transport Layer Security). Προκάτοχος αυτού ήταν το SSL (Secure Socket Layer). Όπως φαίνεται και στην εικόνα 8b το SSL ή TLS εφαρμόζεται στο τέταρτο επίπεδο για να προστατέψει εφαρμογές διαδικτύου, όπως το HTTP, FTP και SMTP.




**Εικόνα 8.** Οι θέση των πρωτοκόλλων ασφαλείας IPsec, SSL/TLS στο TCP/IP πρωτόκολλο (Stallings, W., 2010).

Ένα άλλο πρωτόκολλο ασφαλείας, είναι το IPsec. Το IPsec εφαρμόζεται στο τρίτο επίπεδο του TCP/IP, και χρησιμοποιείται για την προστασία ιδιωτικών δικτύων (Virtual Private Networks – VPN), όπως φαίνεται και από την εικόνα 8a.

# 3.1 Secure Socket Layer (SSL)

Περί τα τέλη του 1995, η εταιρία Netscape δημοσιεύει την έκδοση 3.0 του πρωτοκόλλου ασφαλείας SSL (Secure Socket Layer). Είχαν προηγηθεί άλλες δυο εκδόσεις του. Το 1994 εκδόθηκε η 1.0, παρόλα αυτά η έκδοση αυτή δε δημοσιεύτηκε ποτέ, αλλά χρησιμοποιήθηκε για εσωτερικές ανάγκες της Netscape. Το 1995 εκδόθηκε και δημοσιεύτηκε η έκδοση 2.0. Όμως, τα πολλά κενά ασφαλείας, και οι περιορισμοί που υπήρχαν στην έκδοση αυτή, οδήγησαν σε σύντομο χρονικό διάστημα στην δημοσιοποίηση της τελικής έκδοση 3.0. Το 1996 το SSL περνά στα χέρια της IEFT (Internet Engineering Task Force), η οποία δημιουργεί το TLSgroup με σκοπό τη δημιουργία του Transport Layer (TLS), βάση του SSL v3.0. Το TLS v1.0 έχει αρκετά κοινά με το SSL v3.0 και μπορεί να θεωρηθεί ως η έκδοση 3.1, και θα αναλυθεί σε μεταγενέστερο κεφάλαιο μαζί με όλες τις νέες εκδόσεις του και τις διαφορές τους.

Το πρωτόκολλο SSL δημιουργήθηκε για να παρέχει προστασία από άκρη σε άκρη στην επικοινωνία πελάτη/εξυπηρετητή (client/server), μέσω του διαδικτύου. Λειτουργεί κάτω από το πρωτόκολλο HyperText Transfer Protocol (HTTP), που χρησιμοποιείται για την μεταφορά δεδομένων μέσω του διαδικτύου. Χρησιμοποιεί την ασύμμετρη κρυπτογραφία για την εγκατάσταση ενός ασφαλούς καναλιού επικοινωνίας μεταξύ πελάτη/εξυπηρετητή, με σκοπό την ανταλλαγή κλειδιών συμμετρικής κρυπτογραφίας για την εξασφάλιση της εμπιστευτικότητας όταν θα αρχίσει η ανταλλαγή δεδομένων. Επίσης χρησιμοποιεί την ασύμμετρη για την αυθεντικοποίηση του εξυπηρετητή, η

αυθεντικοποίηση του πελάτη είναι προαιρετική. Τέλος για την εξασφάλιση της ακεραιότητας του μηνύματος, χρησιμοποιεί τον κώδικα αυθεντικοποίησης μηνύματος (MAC).

Το SSL, χωρίζεται σε δυο μέρη. Η σύνοδος (session) αποτελεί στην ουσία το πρώτος μέρος. Η σύνοδος δημιουργείται κατά το πρωτόκολλο της χειραψίας (handshake protocol), και χρησιμοποιείται για τον καθορισμό των κρυπτογραφικών παραμέτρων, που θα χρησιμοποιηθούν κατά την σύνοδο και αργότερα κατά τη σύνδεση. Η σύνοδος αποτελείται από τις ακόλουθες παραμέτρους:

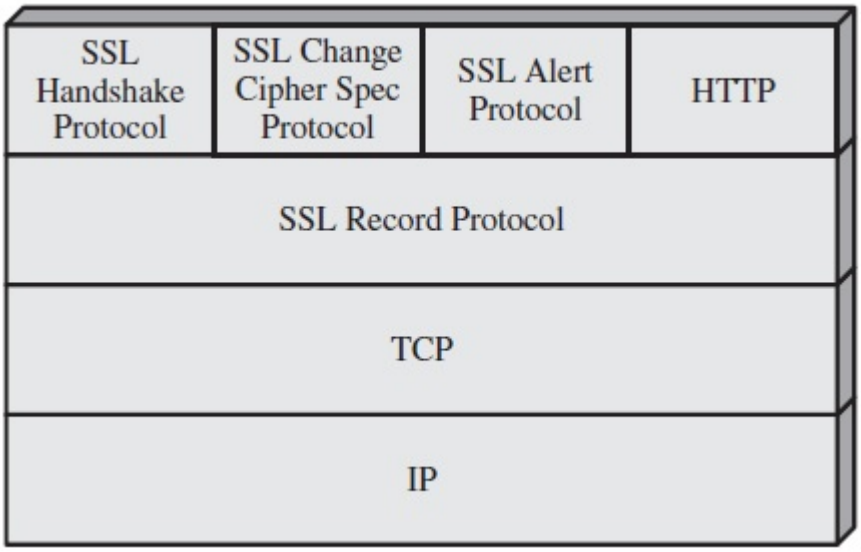
- **Session Identifier:** Είναι μια ακολουθία από byte, που χρησιμοποιείται από το εξυπηρετητή για να καθορίσει αν μια σύνοδος είναι ενεργή ή πρέπει να επανεκκίνηση.
- **Peer Certificate:** Η τιμή αυτή καθορίζει αν θα ζητηθεί από το χρήστη ένα πιστοποιητικό X509.v3. Η τιμή αυτή σχεδόν πάντα είναι μηδέν.
- **Compression Method:** Συνήθως γίνεται συμπίεση των δεδομένων πριν την κρυπτογράφηση.
- **Cipher Spec:** Καθορίζει ποιος αλγόριθμος θα χρησιμοποιηθεί για την κρυπτογράφηση του κυρίου όγκου των δεδομένων (π.χ. AES, DES, κτλ.), τη συνάρτηση κατακερματισμού (π.χ. MD5, SHA-1, κτλ.), και άλλες παραμέτρους, όπως το μέγεθος των δεδομένων της συνάρτησης κατακερματισμού (hash_size).
- **Master Secret:** Ένα διαμοιραζόμενο κλειδί μήκους 48-bytes.
- **Is Resumable:** Είναι μια λογική (Boolean) παράμετρος που καθορίζει αν μια σύνοδος μπορεί να χρησιμοποιηθεί για την εγκαθίδρυση καινούργιων συνδέσεων (ως συνέχεια των προηγουμένων).

Το δεύτερο μέρος, είναι το μέρος της σύνδεσης (connection), όπου πραγματοποιείται η κρυπτογραφημένη μεταφορά δεδομένων. Κάθε σύνδεση είναι παροδική και χαρακτηρίζεται από την σύνοδο. Η σύνδεση καθορίζεται από τις παρακάτω παραμέτρους:

- **Server and Client Random:** Μια ακολουθία από bytes που επιλέγονται από τον πελάτη και τον εξυπηρετητή και είναι διαφορετικά για κάθε σύνδεση.

- **Server Write MAC secret:** Το μυστικό κλειδί που χρησιμοποιείται από το εξυπηρετητή για τα δεδομένα που στέλνονται με κώδικα αυθεντικοποίησης μηνύματος.
- **Client Write MAC secret:** Το μυστικό κλειδί που χρησιμοποιείται από το πελάτη για τα δεδομένα που στέλνονται με κώδικα αυθεντικοποίησης μηνύματος.
- **Server Write Key:** Το συμμετρικό κλειδί που χρησιμοποιείται για την κρυπτογράφηση δεδομένων από τον εξυπηρετητή και την αποκρυπτογράφηση από τον πελάτη.
- **Client Write Key:** Το συμμετρικό κλειδί που χρησιμοποιείται για την κρυπτογράφηση δεδομένων από τον πελάτη και την αποκρυπτογράφηση από τον εξυπηρετητή.
- **Initialization Vectors:** Καθορισμός του διανύσματος αρχικοποίησης όταν χρησιμοποιείται κάποιος ρυθμός λειτουργίας που χρειάζεται διάνυσμα αρχικοποίησης (π.χ. CBC).
- **Sequence Number:** Κάθε πλευρά διατηρεί ξεχωριστό, διαφορετικό αριθμό ακολουθίας (sequence number) για τα πακέτα που έχει παραλάβει, αλλά και έχει στείλει. Ο αριθμός αυτός δεν πρέπει να ξεπερνά το $2^{64}$-1, ειδάλλως μηδενίζεται. Ο αριθμός αυτός μηδενίζεται, επίσης, και στην περίπτωση που σταλεί ή ληφθεί cipher spec.

Η αρχιτεκτονική του πρωτόκολλου SSL φαίνεται στην εικόνα 9, και στην ουσία δεν είναι ένα πρωτόκολλο ενός επιπέδου, αλλά δυο. Το SSL Record Protocol, βρίσκεται στο κάτω επίπεδο, παρέχοντας υπηρεσίες ασφαλείας στα πρωτόκολλα που βρίσκονται στο πιο πάνω επίπεδο. Παρατηρούμε ότι μαζί με το πρωτόκολλο επικοινωνίας HTTP, βρίσκονται στο ίδιο επίπεδο, τα πρωτόκολλα Handshake, Change Cipher Spec και Alert.




**Εικόνα 9.** Αρχιτεκτονική του πρωτοκόλλου SSL (Stallings, W., 2010).

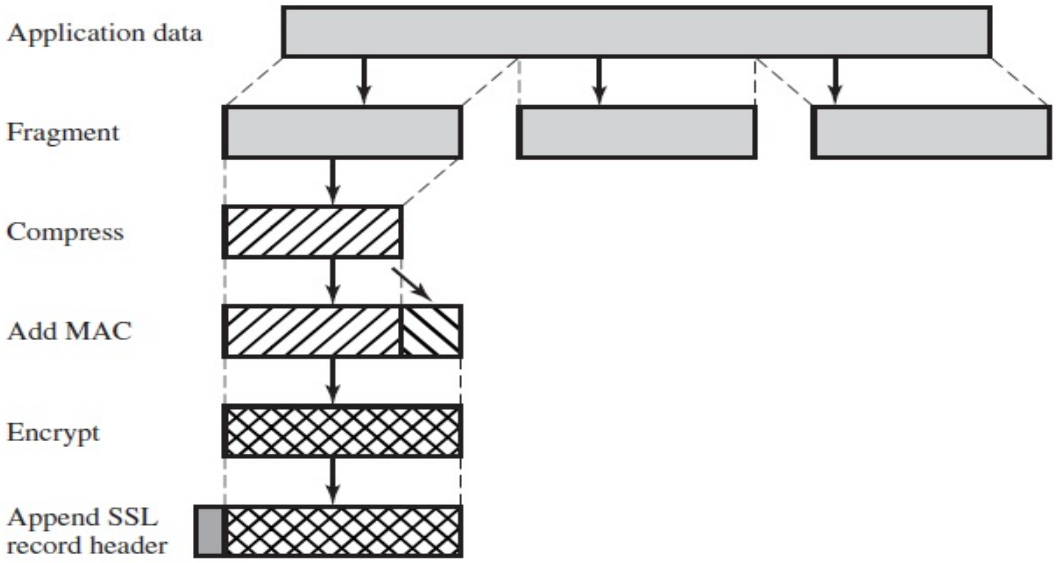
### 3.1.1 SSL Record Protocol

Το SSL Record Protocol προφέρει δυο υπηρεσίες ασφαλείας: Την εμπιστευτικότητα, που επιτυγχάνεται μέσω του μυστικού διαμοιραζόμενου κλειδιού (master shared secret), που παράγεται στο πρωτόκολλο χειραψίας, και χρησιμοποιείται για την κρυπτογράφηση των δεδομένων. Η δεύτερη υπηρεσία είναι η ακεραιότητα μηνύματος, η οποία εξασφαλίζεται από το κώδικα αυθεντικοποίησης μηνύματος και ενός μυστικού διαμοιραζόμενου κλειδιού που ορίζεται στο πρωτόκολλο χειραψίας.

Στην εικόνα 10 βλέπουμε το πώς λειτουργεί το SSL Record Protocol, συγχωνεύοντας τις δυο πιο πάνω υπηρεσίες. Κατ' αρχάς, παίρνει τα δεδομένα, τα τεμαχίζει (fragmentation), στη συνέχεια εφαρμόζει, προαιρετικά, κάποιο αλγόριθμο συμπίεσης (compression) (από το SLL v3.0, και άρα και το TLS v1.0, δεν χρησιμοποιείται κάποιος αλγόριθμος συμπίεσης και η τιμή του πεδίου αυτού είναι μηδέν). Μετά εφαρμόζεται ο κώδικας αυθεντικοποίησης μηνύματος (MAC) και προστίθεται στο τέλος του πακέτου, που ακολούθως κρυπτογραφείται με κάποιο αλγόριθμο συμμετρικής κρυπτογράφησης. Τέλος προστίθεται μια επικεφαλίδα με τις εξής πληροφορίες:

- **Content Type (8-bits):** Που θα προωθηθεί το πακέτο.
- **Major Version (8-bits):** Η μεγαλύτερη έκδοση του SSL, που μπορεί να χρησιμοποιηθεί.

- **Minor Version (8-bits):** Η μικρότερη έκδοση του SSL, που μπορεί να χρησιμοποιηθεί.
- **Compressed Length (16-bits):** Το μήκος των δεδομένων, σε bytes, που έχουν τεμαχιστεί.




**Εικόνα 10.** Η λειτουργία του SSL Record Protocol (Stallings, W., 2010).

### 3.1.2 Change Cipher Spec Protocol

Το πρωτόκολλο αυτό είναι το απλούστερο των τριών πρωτοκόλλων που χρησιμοποιούν το SSL Record Protocol. Το πρωτόκολλο αποτελείται από ένα μήνυμα, μεγέθους ενός byte, και το οποίο καθορίζει αν θα αναθεωρηθούν οι αλγόριθμοι κρυπτογράφησης ή όχι.

### 3.1.3 Alert Protocol

Το Alert Protocol χρησιμοποιείται για να ενημερώσει το χρήστη ότι υπάρχει κάποιο σφάλμα στην προσπάθεια εγκατάστασης ασφαλούς καναλιού επικοινωνίας με τον άλλο συνομιλητή – χρήστη. Το πρωτόκολλο αποτελείται από ένα μήνυμα δυο bytes. Το πρώτο byte παίρνει την τιμή 1, αν το σφάλμα είναι κρίσιμο (fatal), και τερματίζει τη σύνδεση, και την τιμή 2 όταν είναι απλά προειδοποιητικό (warning). Το δεύτερο byte περιέχει ένα κωδικό που παραπέμπει σε κάποιο σφάλμα. Οι κρίσιμοι κώδικες είναι:

- **unexpected message:** Έχει ληφθεί κάποιο μήνυμα που δεν έπρεπε να έχει ληφθεί.
- **bad-record_mac:** Έχει ληφθεί λάθος MAC.
- **decompression_failure:** Η είσοδος της συνάρτησης αποσυμπίεσης είναι εσφαλμένη.
- **handshake_failure:** Ο αποστολέας δεν μπορεί να διαπραγματευτεί της παραμέτρους ασφαλείας που του προτείνονται.
- **illegal_parameter:** Κάποιο μήνυμα χειραψίας είτε δεν εντός χρονικών ορίων, είτε δεν είναι συμβατό με τα άλλα μηνύματα.

Οι κώδικες που δεν είναι κρίσιμοι είναι:

- **close_notify:** Ο αποστολέας ειδοποιεί τον παραλήπτη με αυτό το μήνυμα ότι δεν θα του στείλει άλλα μηνύματα.
- **no_certificate:** Στέλνεται σε περίπτωση που το πιστοποιητικό που ζητήθηκε δεν είναι διαθέσιμο.
- **bad_certificate:** Το πιστοποιητικό που λήφθηκε είναι παρεφθαρμένο.
- **unsupported_certificate:** Ο τύπος του πιστοποιητικού δεν υποστηρίζεται.
- **certificate_revoke:** Το πιστοποιητικό έχει αποσυρθεί από αυτόν που το υπέγραψε.
- **certificate_expired:** Το πιστοποιητικό έχει λήξει.
- **certificate_unknown:** Το πιστοποιητικό είναι άγνωστο και μη αποδεκτό.

### 3.1.4 Handshake Protocol

Το Handshake Protocol (πρωτόκολλο χειραψίας) πραγματοποιείται πριν την ανταλλαγή οποιονδήποτε δεδομένων. Χρησιμοποιείται για την αυθεντικοποίηση του εξυπηρετητή και πελάτη, καθώς και για την επιλογή των κρυπταλγορίθμων που θα χρησιμοποιηθούν για την κρυπτογράφηση και τον κώδικα αυθεντικοποίησης μηνύματος. Τα μηνύματα αποτελούνται από τρία πεδία όπως περιγράφονται παρακάτω:

- **Type (1 byte):** Χρησιμοποιείται για να υποδηλώσει κάποιο από τα δέκα μηνύματα που φαίνονται στην εικόνα 11.
- **Length (3 bytes):** Το μέγεθος του μηνύματος σε bytes.

- **Content (≥ 0 bytes):** Οι παράμετροι του κάθε μηνύματος, όπως φαίνονται και στην εικόνα 11.

| Message Type | Parameters |
|---|---|
| hello_request | null |
| client_hello | version, random, session id, cipher suite, compression method |
| server_hello | version, random, session id, cipher suite, compression method |
| certificate | chain of X.509v3 certificates |
| server_key_exchange | parameters, signature |
| certificate_request | type, authorities |
| server_done | null |
| certificate_verify | signature |
| client_key_exchange | parameters, signature |
| finished | hash value |

**Εικόνα 11.** Οι τύποι των μηνυμάτων στο πρωτόκολλο χειραψίας (Stallings, W., 2010).

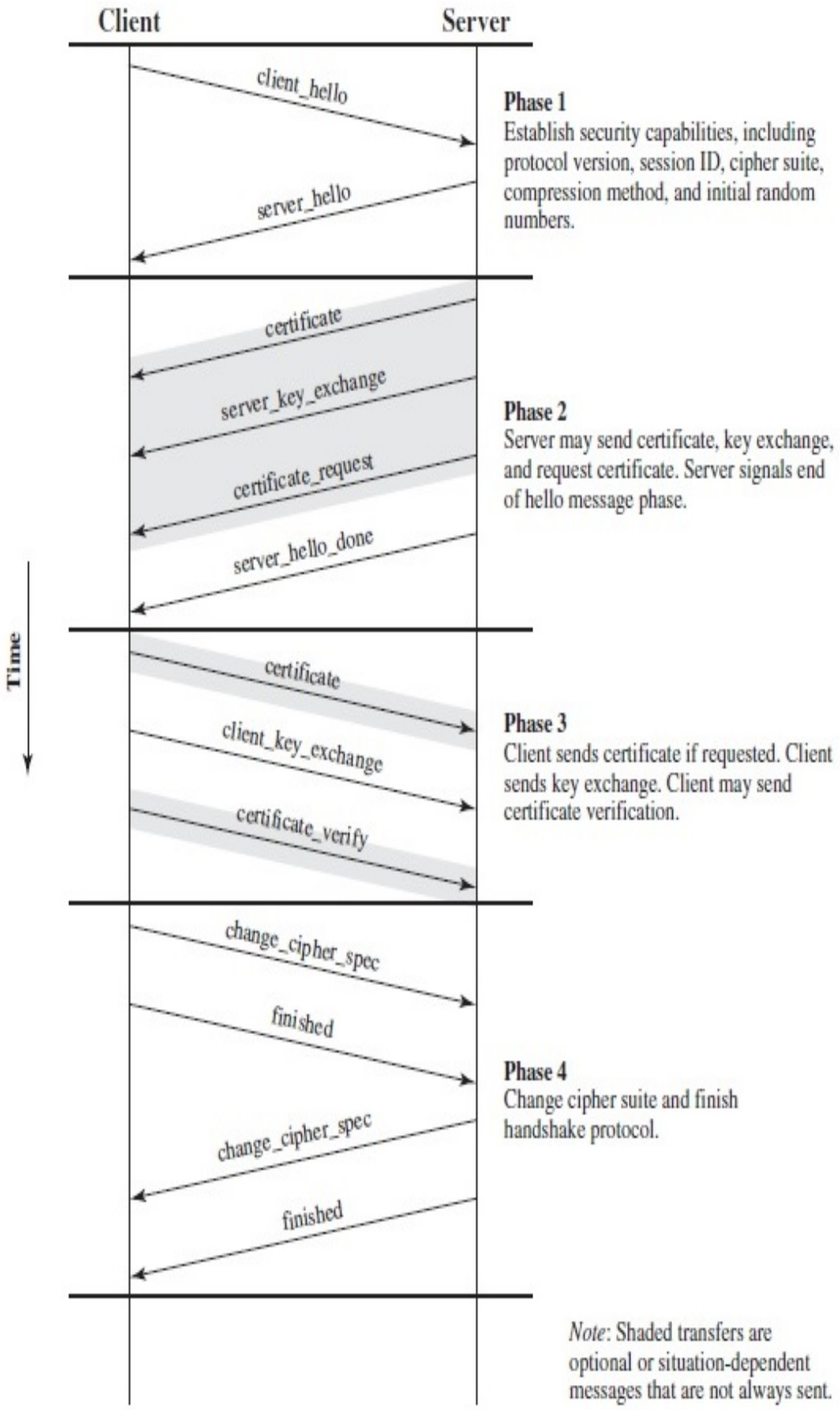
Το πρωτόκολλο χειραψίας χωρίζεται σε τέσσερις φάσεις όπως φαίνεται και στην εικόνα 12. Σε κάθε φάση ανταλλάσσονται κάποια μηνύματα, που μέσω αυτών γίνονται οι επιλογές των παραμέτρων ασφαλείας που θα χρησιμοποιηθούν. Κάποια από τα μηνύματα μπορεί να παραληφθούν αναλόγως των παραμέτρων ασφαλείας που θα επιλεγούν.

a) **Φάση 1: Εγκαθίδρυση Παραμέτρων Ασφαλείας (Establish Security Parameters)**

Η φάση αυτή ξεκινά με το μήνυμα από το πελάτη προς τον εξυπηρετητή client_hello, και σε αυτή αρχικοποιούνται και επιλέγονται οι παράμετροι ασφαλείας που θα χρησιμοποιηθούν. Το μήνυμα client_hello που στέλνεται από τον πελάτη, περιέχει μερικές παραμέτρους για να ξεκινήσει η διαπραγμάτευση. Είναι η παράμετρος Version, που είναι η έκδοση του πρωτοκόλλου ασφαλείας που υποστηρίζει. Επιπλέον επιλέγει και στέλνει ένα τυχαίο αριθμό (nonce), που χρησιμοποιείται για δυο σκοπούς. Πρώτον για την αποτροπή επιθέσεων επανεκτέλεσης μηνυμάτων (replay attacks), και δεύτερο, χρησιμοποιείται στον υπολογισμό των κλειδιών. Η παράμετρος Session ID, είναι μηδενική αν ο πελάτης για πρώτη φορά προσπαθεί να επικοινωνήσει με τον

εξυπηρετητή, ενώ παίρνει κάποια τιμή αν ο πελάτης έχει ξανά επικοινωνήσει με τον εξυπηρετητή, και η σύνδεση μεταξύ τους είχε παραμείνει ανενεργή για κάποιο διάστημα και έχει διακοπεί προσωρινά. Στο CipherSuite καθορίζει με σειρά προτεραιότητας ποιους αλγορίθμους επιθυμεί για κρυπτογράφηση (ασύμμετρη και συμμετρική), και σε συνδυασμό με ποια συνάρτηση κατακερματισμού. Τέλος με το Compression Method, στέλνεται ποια συνάρτηση συμπίεση υποστηρίζει ο πελάτης.

Αφού ο πελάτης στείλει το client_hello μήνυμα, περιμένει ως απάντηση από το εξυπηρετητή το server_hello, το οποίο περιέχει τις ίδιες παραμέτρους με το client_hello. Για την παράμετρο Version ο εξυπηρετητής επιλέγει τη μέγιστη δυνατή που υποστηρίζει αυτός και ο πελάτης. Επιλέγει και αυτός τον δικό του τυχαίο αριθμό (nonce) και ελέγχει την παράμετρο Session ID. Αν αυτή είναι μηδέν, τότε δίδει μια τιμή σ' αυτή, αν έχει κάποια τιμή τότε διατηρεί την ίδια τιμή. Ακολούθως ο εξυπηρετητής επιλέγει τον πιο δυνατό συνδυασμό από τους αλγορίθμους κρυπτογράφησης και τις συναρτήσεις κατακερματισμού που προτείνονται από το πελάτη. Τέλος επιλέγει μια από τις συναρτήσεις συμπίεσης που του έχουν προταθεί.




**Εικόνα 12.** Οι φάσεις του πρωτοκόλλου χειραψίας (Stallings, W., 2010).

Όπως είπαμε η παράμετρος CipherSuite περιέχει ένα μήνυμα με τον αλγόριθμο ανταλλαγής κλειδιών (key exchange algorithm), αλγόριθμο συμμετρικής κρυπτογράφησής, την συνάρτηση κατακερματισμού (MAC Algorithm) και κάποιες άλλες παραμέτρους.

- **Key Exchange Algorithm:** Είναι ο αλγόριθμος ασύμμετρης κρυπτογραφίας που χρησιμοποιείται για την μετάδοση του μυστικού κλειδιού για τη συμμετρική κρυπτογραφία. Οι αλγόριθμοι ανταλλαγής κλειδιών που υποστηρίζονται είναι:
    - **RSA:** Ο πελάτης υπολογίζει και κρυπτογραφεί το μυστικό κλειδί με το δημόσιο κλειδί του εξυπηρετητή.
    - **Σταθερός (Fixed) Diffie-Hellman:** Ο εξυπηρετητής διαμοιράζεται το Diffie-Hellman δημόσιο του κλειδί μέσω του πιστοποιητικού. Ο πελάτης είναι και αυτός κάτοχος ενός δικού του Diffie-Hellman δημόσιου κλειδιού το οποίο διαμοιράζεται με τον εξυπηρετητή είτε μέσω ενός πιστοποιητικού, είτε μέσω του μηνύματος key exchange. Τέλος βάση του Diffie-Hellman υπολογίζεται ένα σταθερός δημόσιο κλειδί.
    - **Εφήμερος (Ephemeral) Diffie-Hellman:** Η μέθοδος αυτή χρησιμοποιεί προσωρινά κλειδιά. Το δημόσιο κλειδί του Diffie-Hellman υπογράφεται με ένα RSA ή DSS ιδιωτικό κλειδί. Ο πελάτης αποκρυπτογραφεί και επιβεβαιώνει με το κατάλληλο δημόσιο κλειδί. Η μέθοδος αυτή θεωρείται ασφαλέστερη από τις άλλες γιατί χρησιμοποιούνται προσωρινά κλειδιά.
    - **Ανώνυμος (Anonymous) Diffie-Hellman:** Είναι ο βασικός Diffie-Hellman, ο οποίος πραγματοποιεί ανταλλαγή κλειδιών χωρίς αυθεντικοποίηση.
- **Cipher Algorithm:** RC2, RC4, DES, 3DES, DES40, IDEA, Fortezza.
- **MAC Algorithm:** MD5, SHA-1.
- **CipherType:** Με ρυθμό λειτουργίας ή χωρίς.
- **IsExportable:** Σωστό ή λάθος.
- **Hash Size:** 0, 16 για MD5 και 20 για SHA-1 bytes.
- **Key Material:** Ακολουθία από bytes, που χρησιμοποιούνται για τη δημιουργία των write keys.
- **IV size:** Το μέγεθος του διανύσματος αρχικοποίησης για τους ρυθμούς λειτουργίας που χρειάζονται διάνυσμα αρχικοποίησης.

b) **Φάση 2: Αυθεντικοποίηση του Εξυπηρετητή και Ανταλλαγή Κλειδιών (Server Authentication and Key Exchange)**

Η φάση αυτή ξεκινά με τον εξυπηρετητή να στέλνει το υπογεγραμμένο και πιστοποιημένο πιστοποιητικό από κάποια Αρχή Πιστοποίησης στον πελάτη. Στη συνέχεια στέλνει το μήνυμα server_key_exchange. Το μήνυμα αυτό διαφέρει αναλόγως της μεθόδου ανταλλαγής κλειδιών που έχει επιλεγεί. Για το Anonymous Diffie-Hellman ο εξυπηρετητής στέλνει δυο τιμές, ένα πρώτο αριθμό, μαζί με την πρωτεύων ρίζα του αριθμού αυτού, και επιπλέον το δημόσιο κλειδί του Diffie-Hellman. Στην περίπτωση του Ephemeral Diffie-Hellman, το μήνυμα περιέχει μια επιπλέον παράμετρο, την ψηφιακή υπογραφή. Τέλος στην περίπτωση του RSA ο εξυπηρετητής δημιουργεί ένα προσωρινό ζεύγος δημοσίου ιδιωτικού κλειδιού και αποστέλλει μέσω του μηνύματος το δημόσιο κλειδί μαζί με την ψηφιακή υπογραφή. Το μήνυμα αυτό παραλείπεται στη περίπτωση που αλγόριθμος ανταλλαγής κλειδιών είναι ο Fixed Diffie-Hellman ή ο RSA.

Ένας μη αυθεντικοποιημένος εξυπηρετητής, δηλ. ένας εξυπηρετητής που χρησιμοποιεί τον αλγόριθμος ανταλλαγής κλειδιών Anonymous Diffie-Hellman, μπορεί να ζητήσει από τον πελάτη να αυθεντικοποιηθεί μέσω του μηνύματος certificate_request. Τέλος ο εξυπηρετητής στέλνει ένα μήνυμα server_hello_done που υποδηλώνει το τέλος των hello μηνυμάτων και περιμένει την ανταπόκριση από τον πελάτη.

c) **Φάση 3: Αυθεντικοποίηση του Πελάτη και Ανταλλαγή Κλειδιών (Client Authentication and Key Exchange)**

Στη τρίτη φάση, ο πελάτης αποστέλλει το πιστοποιητικό του, αν αυτό του έχει ζητηθεί. Αν του ζητηθεί αλλά ο πελάτης δεν κατέχει πιστοποιητικό, τότε στέλνεται ένα alert μήνυμα no-certificate. Στη συνέχεια, ο πελάτης στέλνει το μήνυμα client_key_exchange. Το μήνυμα αυτό, όπως και προηγουμένως το server_key_exchange, τροποποιείται αναλόγως του αλγόριθμου ανταλλαγής κλειδιών. Για το Anonymous ή Ephemeral Diffie-Hellman, στέλνονται οι παράμετροι του δημοσίου κλειδιού του πελάτη. Για το Fixed Diffie-Hellman στέλνεται κενό μήνυμα, διότι οι παράμετροι έχουν σταλεί με το προηγούμενο μήνυμα, μέσω του πιστοποιητικού. Για το RSA ο πελάτης δημιουργεί ένα 48-byte pre_master_secret, το οποίο κρυπτογραφεί με το προσωρινό δημόσιο κλειδί πριν το στείλει. Τελευταίο μήνυμα της φάσης αυτής είναι το certificate_verify, το οποίο

μπορεί να σταλεί για να επαληθεύσει το πιστοποιητικό που έστειλε ο πελάτης προηγουμένως.

d) **Φάση 4: Τελική Φάση (Finish)**

Στην τέταρτη και τελευταία φάση, ο πελάτης στέλνει το μήνυμα change_cipher_spec σύμφωνα με τους συμφωνημένους αλγορίθμους και κλειδιά. Αμέσως μετά, στέλνει το μήνυμα finished κρυπτογραφημένο, σύμφωνα με τους αλγορίθμους και τα κλειδιά που έχουν προ συμφωνηθεί για να επιβεβαιώσει την ορθότητά τους. Ο εξυπηρετητής, μόλις λάβει τα μηνύματα αυτά, στέλνει ακριβώς τα ίδια μηνύματα. Εδώ τελειώνει το πρωτόκολλο χειραψίας και ξεκινά η κρυπτογραφημένη ανταλλαγή δεδομένων.

Η επιτυχία της εγκαθίδρυσης ενός ασφαλούς καναλιού επικοινωνίας, οφείλεται στην ουσία στον υπολογισμό του μυστικού κλειδιού master_secret. Για τον υπολογισμό του θα πρέπει πρώτα να γίνει ανταλλαγή της ποσότητας pre_master_secret. Η ανταλλαγή πραγματοποιείται μέσω των αλγορίθμων ανταλλαγής κλειδιών και γίνεται με διαφορετικό τρόπο κάθε φορά. Στην περίπτωση του RSA ο πελάτης υπολογίζει το pre_master_secret, το κρυπτογραφεί με το δημόσιο κλειδί του εξυπηρετητή και το στέλνει στον εξυπηρετητή. Στην περίπτωση του Diffie-Hellman, κάθε πλευρά, πελάτης/εξυπηρετητής, υπολογίζει Diffie-Hellman ένα δημόσιο κλειδί. Αφού ανταλλαχθούν, κάθε πλευρά, χρησιμοποιεί το Diffie-Hellman για τον υπολογισμό του pre_master_secret. Εφόσον είναι γνωστή η ποσότητα pre_master_secret και στις δυο πλευρές, τότε το master_secret υπολογίζεται βάση της εξίσωσης 3.1

$$
\begin{aligned}
&master_secret = MD5(pre_master_secret \,||\, SHA('A') \,||\, pre_master_secret \,||\\
&ClientHello.random \,||\, ServerHello.random)) \,||\\
&MD5(pre_master_secret \,||\, SHA('BB') \,||\, pre_master_secret \,||\\
&ClientHello.random \,||\, ServerHello.random)) \,||\\
&MD5(pre_master_secret \,||\, SHA('CCC') \,||\, pre_master_secret \,||\\
&ClientHello.random \,||\, ServerHello.random))
\end{aligned}
$$

(3.1)

όπου,

οι τιμές ClientHello.random και ServerHello.random είναι οι τυχαίες τιμές που ανταλλάσσονται κατά την ανταλλαγή των client_hello και server_hello μηνυμάτων (Freier, A., Karlton, P., Kocher, P., 2011) (Hickman, K., E., B., 1995) (Meyer, C., Schwenk,

J., 2013, σσ. 1-2) (Ristić, I., 2014) (Stallings, W., 2010, σσ. 140-155) (Stallings, W., 2014, σσ. 523-538) (Γεωργιάδου, Μ., Ζιαζίας, Α., 2007, σσ. 63-70) (Κολοκοτρώνης Ν. , 2014) (Μάντζιος, Ι., Μπαντίας, Σ., 2007, σσ. 20-31) (Μουρσελλά, Ε., 2013, σσ. 12-16) (Μπόζης., Ε., Ζ., 2011, σσ. 78-79).

## 3.2 Transport Layer Security (TLS)

Το 1999 το TLSgroup δημοσιεύει την πρώτη έκδοση του πρωτοκόλλου TLS (Transport Layer Security) την 1.0, η οποία ορίζεται στο RFC 2246 (Dierks, T., Allen, C., 1999), και αποτελεί απόγονο του SSL v3.0. Τα δυο πρωτόκολλα δεν έχουν πολλές διαφορές, παρόλα αυτά, δεν υπάρχει συμβατότητα μεταξύ τους. Ένα άλλο εξίσου σημαντικό στοιχείο είναι το γεγονός, ότι το TLS v1.0 δεν επιτρέπει σε μια υλοποίηση, την υποβάθμιση του σε SSL v3.0.

Στη συνέχεια, το 2008, εκδόθηκε η έκδοση TLS v1.1, η οποία περιγράφεται στο RFC 4346 (Dierks, T., Rescorla, E., 2006), και παρουσιάζει σημαντικές διαφορές σε σχέση με την προηγούμενη έκδοση. Οι αλλαγές περιγράφονται παρακάτω:

- Προσθήκη προστασία ενάντια σε επιθέσεις αλγορίθμου ροής λειτουργίας CBC.
- Αντικατάσταση του προκαθορισμένου διανύσματος αρχικοποίησης IV από ένα ρητό διάνυσμα αρχικοποίησης.
- Άλλαξε ο τρόπος διαχείρισης των σφαλμάτων γεμίσματος (padding errors).
- Υποστήριξη παραμέτρων στο IANA (Internet Assigned Numbers Authority).

Δυο χρόνια, μετέπειτα, δημοσιεύθηκε η έκδοση που χρησιμοποιούμε μέχρι σήμερα, η 1.2, και περιγράφεται στο RFC 5246, και είναι βασισμένη στην έκδοση 1.1 με κάποιες διαφοροποιήσεις που περιγράφονται παρακάτω:

- Η συνάρτηση κατακερματισμού SHA256 αντικαθιστά το συνδυασμό MD5 –SHA-1 κατά την επαλήθευση του μηνύματος finished.
- Ο συνδυασμός MD5-SHA-1, στην ψευδοτυχαία συνάρτηση, αντικαταστάθηκε από το SHA-256, με δυνατότητα επιλογής ψευδοτυχαίων συναρτήσεων από μια σουίτα αλγορίθμων.

- Στα ψηφιακά υπογεγραμμένα στοιχεία αντικαθιστάτε ο συνδυασμός MD5-SHA-1, με μια συνάρτηση κατακερματισμού που συμφωνείται στη χειραψία, με προεπιλογή την SHA-1.
- Βελτίωση των δυνατοτήτων του εξυπηρετητή και του πελάτη στην επιλογή κατάλληλων αλγορίθμων για συναρτήσεις κατακερματισμού και ψηφιακές υπογραφές.
- Επέκταση υποστήριξης αλγορίθμων που χρησιμοποιούνται για αυθεντικοποίηση.
- Προτέθηκαν επεκτάσεις του TLS, καθώς και ο κρυπταλγόριθμος AES (Advanced Encryption Standard).
- Οι ειδοποιήσεις (Alerts) πρέπει υποχρεωτικά, χαρακτηρίζονται από ταμπέλα MUST, να στέλνονται.
- Οι αλγόριθμοι συμμετρικής κρυπτογραφίας IDEA και DES αποσύρονται από τις επιλογές.
- Αν ζητηθεί πιστοποιητικό από τον πελάτη μέσω του μηνύματος certificate_request, είναι υποχρεωμένος να στείλει μια άδεια λίστα πιστοποιητικού, αν δεν διαθέτει πιστοποιητικό.

Αργότερα το TLS v1.2 θωρακίστηκε ακόμα περισσότερο, αφού σύμφωνα με το RFC 6176 (Turner, S., Polk, T., 2011), δεν επιτρέπει την υποβάθμιση του πρωτοκόλλου στο ιδιαίτερα επιρρεπή πρωτόκολλο SSL v2.0.

Μετά τις τελευταίες γνωστές επιθέσεις του FREAK, το οποίο οδηγεί τον εξυπηρετητή στην υποβάθμιση της χρήσης κλειδιού στο μήκος των 512 bits (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015) και του Logjam, που υποχρεώνει τον εξυπηρετητή να υποβαθμίσει τον αλγόριθμο ανταλλαγής κλειδιών Diffie-Hellman στο να υποβαθμίσει το κλειδί που χρησιμοποιεί στο μήκος των 512 bits (Adrian, D., Bhargavan, K., Durumeric, Z., Caudry, P., Green, M., Halderman, A., J., Heninger, N., Springall, D., Thome, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanela-Beguelin, S., Zimmerman, P., 2015), είναι υπό μελέτη η καινούργια έκδοση του πρωτοκόλλου TLS v1.3.

Το πρωτόκολλο TLS υποστηρίζει όλους τους κωδικούς ειδοποιήσεων (Alert codes) που υποστηρίζει και το SSL v3.0, εκτός από το no_certificate. Εκτός αυτού νέοι κωδικοί

ειδοποιήσεων που εισάγονται στο TLS και θεωρούνται κρίσιμοι, παρουσιάζονται παρακάτω:

- **record-overflow:** Το μέγεθος του κρυπτογραφημένου κειμένου που επισυνάπτεται στο τέλος του μηνύματος είναι μεγαλύτερο από το μήκος $2^{14}$+2048 bytes, ή το αποκρυπτογραφημένο κείμενο, ξεπερνά το μήκος $2^{14}$+1024 bytes.
- **unknown_ca:** Το πιστοποιητικό που λήφθηκε, είναι κάτω από μια έγκυρη αλυσίδα πιστοποιητικών, παρόλα αυτά, το πιστοποιητικό δεν μπορεί να ταυτοποιηθεί με κάποια αξιόπιστη Αρχή Πιστοποίησης (CA).
- **access_denied:** Ενώ το πιστοποιητικό που έχει ληφθεί είναι έγκυρο, όταν ζητηθεί έλεγχος πρόσβασης (access control), τότε ο εξυπηρετητής αρνείται να διαπραγματευτεί.
- **decode_error:** Το μήνυμα δε μπορεί να αποκωδικοποιηθεί.
- **protocol_version:** Η έκδοση του πρωτοκόλλου που ο πελάτης προσπαθεί να διαπραγματευτεί δεν υποστηρίζεται.
- **insufficient_security:** Όταν ο πελάτης προσπαθεί να διαπραγματευτεί αλγορίθμους λιγότερο ασφαλείς, από αυτούς που υποστηρίζει ο εξυπηρετητής.
- **unsupported_extension:** Όταν ο πελάτης λάβει ένα εκτεταμένο sever_hello μήνυμα που δεν ανταποκρίνεται στο client_hello μήνυμα που έχει στείλει.
- **internal_error:** Όταν συμβεί ένα εσωτερικό λάθος που δεν συνάδει με τον επικοινωνούντα κόμβο, είτε είναι ο πελάτης είτε ο εξυπηρετητής, ή δεν συνάδει με την ορθότητα του πρωτοκόλλου και οδηγεί στο κλείσιμο της σύνδεσης.
- **decrypt_error**: Αυτό συμβαίνει είτε στην περίπτωση της επαλήθευσης της ψηφιακής υπογραφής, είτε στην αποκρυπτογράφηση του κλειδιού, είτε στην επαλήθευση του μηνύματος finished.

Τέλος υπάρχουν και δυο κωδικοί ειδοποίησης, οι οποίοι δεν είναι κρίσιμοι:

- **user_canceled:** Η χειραψία έχει τερματιστεί για κάποιο λόγω που δεν έχει σχέση με το πρωτόκολλο.
- **no_renegotiation:** Η προειδοποίηση αυτή στέλνετε μετά από hello μήνυμα, όταν μια από τις δυο πλευρές δεν υποστηρίζει τη διαπραγμάτευση των παραμέτρων

ασφαλείας. Παρόλο που το σφάλμα είναι πολύ σημαντικό, εντούτοις το μήνυμα είναι απλώς προειδοποιητικό και δεν τερματίζει την σύνδεση.

Ακόμα μια διαφορά του πρωτοκόλλου TLS από το SSL είναι ο τερματισμός της υποστήριξης κάποιων αδυνάμων κρυπταλγορίθμων. Για παράδειγμα η αφαίρεση του Fortezza από τους αλγορίθμους ανταλλαγής κλειδιών, και επιπλέον η αφαίρεση του από τους αλγορίθμους συμμετρικής κρυπτογραφίας.

Τέλος, μια ακόμα σημαντική διαφορά του TLS από το SSL είναι ο τρόπος υπολογισμού της μυστικής ποσότητας master_secret. Ενώ το pre_master_secret υπολογίζεται με τον ίδιο ακριβώς τρόπο, όπως στο SSL, στο TLS το master_secret υπολογίζεται από μια συνάρτηση κατακερματισμού του pre_master_secret και τις ψευδοτυχαίες τιμές που περικλείονται στα hello μηνύματα, και περιγράφεται στην σχέση 3.2.

$$master_secret = PRF(pre_master_secret, "master_secret", ClinetHello.random \| ServerHello.random)$$

(3.2)

Επιπλέον το key_block, που χρησιμοποιείται στα κρυπτογραφικά κλειδιά του κώδικα του μηνύματος αυθεντικοποίησης (MAC), στα κλειδιά της συνόδου (session keys) και στα διανύσματα αρχικοποίησης (IVs), δεν υπολογίζεται συναρτήσει του master_secret αλλά από τη σχέση 3.3 (Adrian, D., Bhargavan, K., Durumeric, Z., Caudry, P., Green, M., Halderman, A., J., Heninger, N., Springall, D., Thome, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanela-Beguelin, S., Zimmerman, P., 2015, pp. 1-3) (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015, pp. 1-2) (Dierks, T., Allen, C., 1999) (Dierks, T., Rescorla, E., 2006) (Dierks, T., Rescorla, E., 2008) (Jager., T., Kholar., F., Schage., S., Schwenk, J., 2013, pp. 11-14) (Ristić, I., 2014) (Stallings, W., 2010, pp. 156-160) (Stallings, W., 2014, pp. 539-543) (Μπόζης., Ε., Ζ., 2011, pp. 78-83) (Μωραΐτης, Δ., Σ., 2012, pp. 11-12).

$$key_block = PRF(master_secret, "key_expansion", SecurityParametres.server_random \| SecurityParametres.client_random)$$

(3.3)

## 3.3 IPsec

Στο τρίτο επίπεδο του TCP/IP πρωτοκόλλου λειτουργεί το επίπεδο του Δικτύου, το επίπεδο IP, το οποίο δεν προσφέρει καμία αυθεντικοποίηση, μιας και η συνάρτηση που πραγματοποιεί έλεγχο σφαλμάτων είναι γνωστή, και οποιαδήποτε αλλαγή στο μήνυμα μπορεί να μην είναι ανιχνεύσιμη με κατάλληλη τροποποίηση και του μηνύματος διόρθωσης σφάλματος που προσαρτάται στο τέλος του μηνύματος. Επιπλέον δεν προσφέρει καθόλου εμπιστευτικότητα, διότι τα δεδομένα, μεταφέρονται σε απλό κείμενο, επομένως, οποιοσδήποτε μπορεί να τα διαβάσει.

Λύση στις πιο πάνω αδυναμίες έρχεται να δώσει το πρωτόκολλο ασφαλείας IPsec, το οποίο προστατεύει τα δεδομένα στο επίπεδο του Δικτυού, και χρησιμοποιείται ευρέως σε εφαρμογές Εικονικών Δικτύων (Virtual Private Networks (VPNs)). Λόγω του γεγονότος ότι εφαρμόζεται στο επίπεδο 3 δε χρειάζεται κάποιο επιπλέον λογισμικό όπως συνέβαινε στο SSL/TLS. Το πρωτόκολλο έχει σχεδιαστεί για το νέο πρωτόκολλο IPv6, παρόλα αυτά, είναι συμβατό και με το πρωτόκολλο IPv4.

Το πρωτόκολλο IPsec δεν είναι ένα απλό πρωτόκολλο, αλλά εμπεριέχει άλλα πρωτόκολλα για τη λειτουργία του. Το πρωτόκολλο Authentication Header (AH) που χρησιμοποιείται για αυθεντικοποίηση μόνο, το Encapsulating Security Payload (ESP), που χρησιμοποιείται για την εξασφάλιση της εμπιστευτικότητας, αλλά μπορεί να παρέχει και ταυτόχρονα αυθεντικοποίηση και τέλος το Internet Key Exchange (IKE), που χρησιμοποιείται για ανταλλαγή κλειδιών και καθορισμό των παραμέτρων ασφαλείας.

Το σύνολο των κλειδιών που θα χρησιμοποιηθούν σε επικοινωνία μεταξύ δυο κόμβων, που προστατεύεται από το IPsec, καθώς και οι παράμετροι ασφαλείας που θα χρησιμοποιηθούν και ποια επιπλέον πρωτόκολλα (AH ή ESP) καταγράφονται σε ένα συσχετισμό ασφαλείας (Security Association (SA)). Ο συσχετισμός ασφαλείας περικλείει ένα πρωτόκολλο κάθε φορά AH ή ESP. Μπορεί να γίνει εφαρμογή και των δυο πρωτοκόλλων σε μια σύνδεση αφού επιτρέπεται ο καθορισμό περισσότερου από ενός συσχετισμού ασφαλείας. Ο καθορισμός είναι αμφίδρομος, πρέπει να καθοριστεί και στους δυο επικοινωνούντες κόμβους, και μπορεί να καθοριστεί χειροκίνητα, μπορεί να εφαρμοστεί στις περιπτώσεις μικρών δικτυών, ή αυτόματα, συνίσταται για μεγάλα

δίκτυα. Οι συσχετισμοί ασφαλείας καθορίζουν τις επιλογές των παραμέτρων για την περίπτωση που ο κόμβος λειτουργεί ως αποστολέας (outbound) και στην περίπτωση που λειτουργεί ως παραλήπτης (inbound).

Ένας συσχετισμός ασφαλείας, περιγράφεται από τρεις παραμέτρους. Πρώτον το Security Parameter Index (SPI), το οποίο είναι ένας αριθμός μήκους 32-bit, που καθορίζει αν ο συσχετισμός ασφαλείας είναι για αποστολέα ή παραλήπτη. Το SPI είναι συσχετισμένο με την βάση συσχετισμών ασφαλείας. Δεύτερη παράμετρος είναι η IP διεύθυνση αποστολής, που μπορεί να είναι ένα τερματικός σταθμός ή ένα σύστημα δικτύωσης, όπως δρομολογητής ή πύλη ασφαλείας. Τελευταία παράμετρος είναι το πρωτόκολλο που χρησιμοποιείται AH ή ESP.

Για τον έλεγχο και τη λειτουργία του IPsec χρησιμοποιούνται δυο βάσεις δεδομένων. Η μια είναι η Security Association Database (SAD), που διατηρεί μια βάση με τις ενεργές συσχετίσεις ασφαλείας και χρησιμοποιείται για επεξεργασία των εισερχόμενων πακέτων (inbound). Κάθε τερματικός σταθμός ή σύστημα δικτύωσης διατηρεί μια τέτοια βάση. Η άλλη βάση δεδομένων είναι η Security Policy Database (SPD), και καθορίζει ποια ή ποιες συσχετίσεις ασφαλείας θα εφαρμοστούν. Χρησιμοποιείται στην περίπτωση των εξερχόμενων πακέτων (outbound) και όπως και πριν κάθε τερματικό ή σύστημα δικτύωσης διατηρούν μια τέτοια βάση. Τα πακέτα σε κάθε περίπτωση ελέγχου μπορεί να απορριφθούν, αν δεν μπορούν να γίνουν αποδεκτές οι παράμετροι τους, να προσπελάσουν χωρίς καμία επιπλέον εφαρμογή παραμέτρων ασφάλειας και τέλος να προσπελάσουν εφόσον εφαρμοστούν διάφοροι παράμετροι ασφαλείας.

Η βάση δεδομένων SAD περιέχει μια παράμετρο που ονομάζεται Sequence Number. Ο αριθμός αυτός αρχικά έχει την τιμή 0, και χρησιμοποιείται για προστασία από τις επιθέσεις επανεκτέλεσης μηνυμάτων (replay attacks). Ο αριθμός αυτός, αυξάνεται κατά ένα κάθε φορά που ένα πακέτο στέλνεται. Έτσι ο παραλήπτης γνωρίζει αν το πακέτο που έφτασε σ' αυτόν έχει ληφθεί ή όχι βάση του Sequence Number. Πακέτα που έχουν Sequence Number ίσο με πακέτα που έχουν ήδη ληφθεί, απορρίπτονται, ειδάλλως γίνονται αποδεκτά. Όταν ο Sequence Number φτάσει σε ένα ορισμένο μέγεθος, συνήθως $2^{32}$-1, τότε μηδενίζεται και ζητείται ξανά επανακαθορισμός των συσχετισμών ασφαλείας.

### 3.3.1 Το Πρωτόκολλο Authentication Header (AH)

Το πρωτόκολλο Authentication Header προσφέρει αυθεντικοποίηση της προέλευσης των δεδομένων. Ο στόχος του είναι η εξασφάλιση της ακεραιότητας και της απρόσκοπτης μεταφοράς των πακέτων, χωρίς την τροποποίηση τους κατά τη μεταφορά. Το AH προστατεύει τα δεδομένα που μεταφέρονται στο IP επίπεδο μαζί με της επικεφαλίδες (Headers). Δεν παρέχει προστασία σε τμήματα του πακέτου που τροποποιούνται κατά τη μεταφορά του, όπως για παράδειγμα το Time To Live (TTL).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Payload Len  |          RESERVED             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Security Parameters Index (SPI)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Sequence Number Field                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                Integrity Check Value-ICV (variable)           |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

**Εικόνα 13.** Η διάταξη του πακέτου του Authentication Header (Kent S. , 2005α).

Το AH περιγράφεται στο RFC 4302 (Kent S. , 2005α), και στην εικόνα 13 φαίνεται η δομή του πακέτου. Το πεδίο Next Header είναι αριθμός, που έχει μήκος 8-bit, και μας περιγράφει τι θα ακολουθήσει μετά (π.χ. TCP = 6, UDP = 17, κτλ.). Είναι το ανώτατο πεδίο που προστατεύει το AH. Το επόμενο πεδίο είναι αυτό του Payload Len, το οποίο είναι και αυτό μήκους 8-bit και εκφράζει το μήκος του πακέτου AH μείον δυο, και παρουσιάζεται σε μονάδες των τεσσάρων οκτάδων (4-octets). Το πεδίο RESERVED, προτίθεται να χρησιμοποιηθεί για μελλοντική χρήση, και προς το παρόν περιέχει μόνο μηδενικά. Επόμενο είναι το πεδίο Security Parameters Index (SPI), όπως περιγραφικέ και πιο πάνω είναι μια τιμή άμεσα συνδεδεμένη με την εφαρμογή του κατάλληλου συσχετισμού ασφαλείας, βάση της διεύθυνσης IP του αποστολέα.

Στη συνέχεια είναι το πεδίο Sequence Number Field. Το πεδίο έχει μήκος 32-bit και περιέχει ένα αριθμό που αυξάνεται κάθε φορά που στέλνεται ένα πακέτο, και χρησιμοποιείται για προστασία από τις επιθέσεις επανεκτέλεσης μηνυμάτων (replay

attacks), όπως περιγράφεται και πιο πάνω. Τέλος είναι το πεδίο Integrity Check Value (ICV), που έχει μεταβλητό μήκος και είναι ακέραιο πολλαπλάσιο των 32-bit για IPv4 και των 64-bit για IPv6, και χρησιμοποιείται για τον έλεγχο της ορθότητας του μηνύματος.

### 3.3.2 Το Πρωτόκολλο Encapsulating Security Payload (ESP)

Το πρωτόκολλο Encapsulating Security Payload μπορεί να παρέχει εμπιστευτικότητα, μέσω της κρυπτογράφησης των δεδομένων, και προστασία από επιθέσεις επανεκτέλεσης μηνυμάτων. Επιπλέον μπορεί να παρέχει και αυθεντικοποίηση μηνυμάτων. Σ' αντίθεση με το AH το ESP δεν προστατεύει την επικεφαλίδα της διεύθυνσης IP, αφήνοντας την εκτεθειμένη σε κοινή θέα, και έτσι ένας επιτιθέμενος απλά με την ανάλυση κίνησης (traffic analysis) να μπορεί να προσδιορίσει από προέρχονται και που πηγαίνουν τα πακέτα. Παρόλα αυτά, δεν μπορεί να προσδιορίσει τι είδους εφαρμογής είναι τα πακέτα (MIME, FTP, κτλ.), ούτε τι περιέχεται μέσα στα πακέτα, διότι το περιεχόμενό τους είναι κρυπτογραφημένο.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ----
|               Security Parameters Index (SPI)                 | ^Int.
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Cov-
|                      Sequence Number                          | |ered
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ | ----
|                    Payload Data* (variable)                   | |   ^
~                                                               ~ |   |
|                                                               | |Conf.
+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Cov-
|               |     Padding (0-255 bytes)                     | |ered*
+-+-+-+-+-+-+-+-+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |   |
|                               |  Pad Length   | Next Header   | v   v
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ------
|         Integrity Check Value-ICV   (variable)                |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

**Εικόνα 14.** Η διάταξη του πακέτου του Encapsulating Security Payload (Kent S. , 2005β).

Το πρωτόκολλο ESP, περιγράφεται στο RFC 4303 (Kent S. , 2005β), και η δομή του πακέτου του φαίνεται στην εικόνα 14. Το πεδίο Security Parameters Index, χρησιμοποιείται για τον καθορισμό των συσχετίσεων ασφαλείας που θα εφαρμοστούν. Το Sequence Number, χρησιμοποιείται για προστασία από τις επιθέσεις επανεκτέλεσης μηνυμάτων.

Στη συνέχεια, το πεδίο Payload Data περιέχει τα δεδομένα του πακέτου που προστατεύονται. Το είδος των δεδομένων που προστατεύονται καθορίζεται από το Next Header. Ακολούθως, το πεδίο Padding παίρνει μια τιμή από 0-255 οκτάδες, και χρησιμοποιείται για να επιμηκύνει το μήκος του Payload Data, ώστε να ταιριάζει με το μήκος των τμήμα που κρυπτογραφούνται, και για να ευθυγραμμίσει το επόμενο πεδίο. Το Pad Length, έχει μήκος 8-bits, και λαμβάνει μια τιμή μεταξύ 0-255 που αντιστοιχεί με το μέγεθος των bits που προστίθενται στο Padding. Το επόμενο πεδίο έχει μήκος 8-bits, ονομάζεται Next Header και είναι ένας αριθμός που περιγράφει τι δεδομένα περιέχονται στο πακέτο (π.χ. TCP = 6, UDP = 17, κτλ.). Τέλος το πεδίο Integrity Check Value έχει μήκος πολλαπλάσιο των 32-bit, και χρησιμοποιείται για τον έλεγχο τροποποίησης του μηνύματος κατά την μεταφορά.

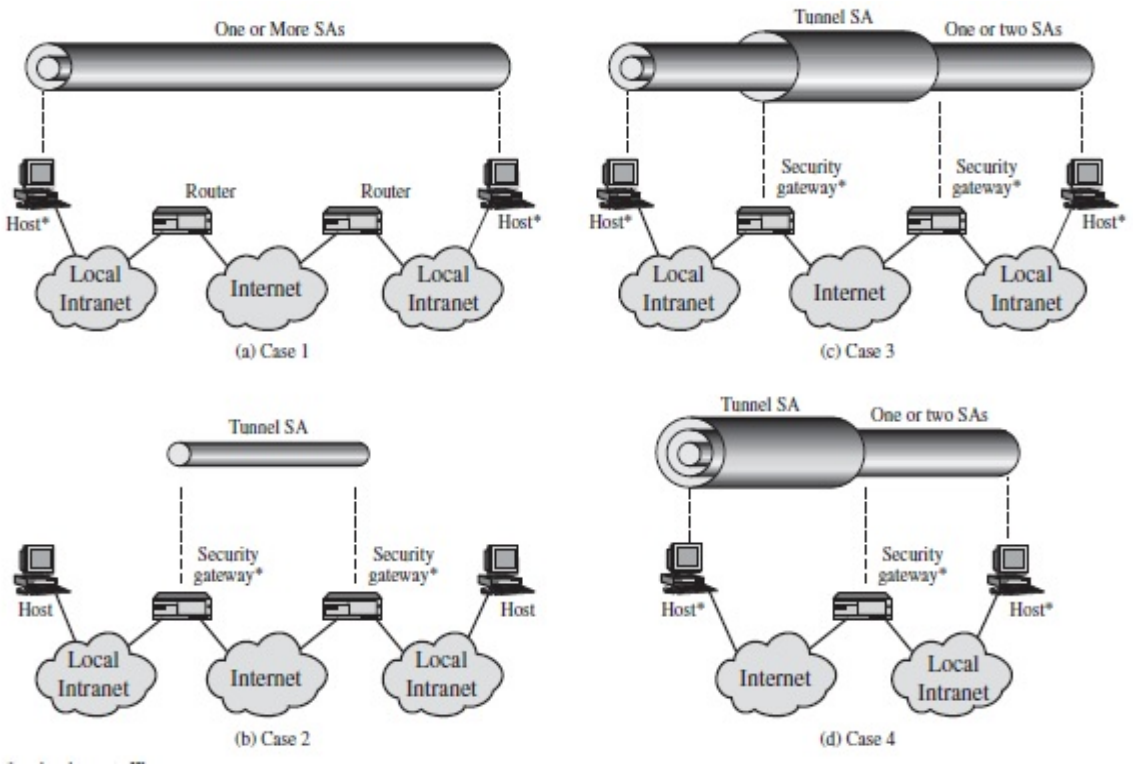
### 3.3.3 Transport and Tunnel Mode

Υπάρχουν δυο τρόποι με τους οποίους μπορεί να λειτουργήσει το IPsec. Και τα δυο πρωτόκολλα AH και ESP μπορούν να λειτουργήσουν και στους δυο τρόπους λειτουργίας. Το Transport Mode χρησιμοποιείται περισσότερο σε περιπτώσεις που θέλουμε να διαφυλάξουμε την επικοινωνία απ' άκρη σ' άκρη, δηλαδή χρήστη προς χρήστη. Προστατεύει τα δεδομένα ανώτερων επιπέδων επικοινωνίας, όπως είναι το TCP ή UDP, αλλά δε προστατεύει τη επικεφαλίδα της διεύθυνσης IP (IP Header).

Το Tunnel Mode χρησιμοποιείται για να προστατέψει την επικοινωνία μεταξύ συσκευών δικτύωσης, όπως δρομολογητές, πύλες ασφαλείας κτλ. Το Tunnel Mode προστατεύει και την επικεφαλίδα της διεύθυνσης IP, αφού μετά την εφαρμογή του πρωτοκόλλου AH ή ESP στο πακέτο, αυτό ενθυλακώνεται σε ένα νέο πακέτο με καινούργια διεύθυνση IP. Η δρομολόγηση του πακέτου γίνεται βάση της εξωτερικής, καινούργιας διεύθυνσης IP.

Στην εικόνα 15 φαίνονται τέσσερις διαφορετικές περιπτώσεις που μπορεί να χρησιμοποιηθούν είτε το Transport Mode, είτε το Tunnel Mode είτε ο συνδυασμός τους. Στη πρώτη περίπτωση (case 1) είναι μια χρήστη προς χρήστη επικοινωνία και μπορεί να προστατευτεί με τέσσερις διαφορετικούς συνδυασμούς. Πρώτο με το AH στο Transport Mode. Δεύτερο με το ESP στο Transport Mode. Τρίτο με το ESP

προστατευόμενο από το AH στο Transport Mode. Τέλος οποιαδήποτε από τις πιο πάνω περιπτώσεις ενθυλακωμένη σε AH ή ESP σε Tunnel Mode. Στη δεύτερη περίπτωση (case 2) η επικοινωνία μεταξύ δυο συσκευών δικτύωσης μπορεί να καλύπτεται από το πρωτόκολλο AH, ή το ESP, ή το ESP με αυθεντικοποίηση σε λειτουργία Tunnel Mode. Στην τρίτη περίπτωση (case 3), αποτελεί ένα συνδυασμό των δυο προηγουμένων περιπτώσεων. Στην ουσία η επικοινωνία μέσω των δυο συσκευών δικτύωσης καλύπτεται σύμφωνα με την περίπτωση 2, ενώ η διασφάλιση της επικοινωνίας από χρήστη σε χρήστη μπορεί να καλυφθεί με ένα πλήθος δυνατών συνδυασμών που παρουσιάστηκαν στην περίπτωση 1, προσφέροντας αυθεντικοποίηση ή εμπιστευτικότητα ή ενίοτε και τα δυο μαζί. Τέλος, η τέταρτη περίπτωση (case 4), είναι η περίπτωση όπου ένας χρήστης συνδέεται σε μια απομακρυσμένη μηχανή μέσω του διαδικτύου. Για τη σύνδεση του χρήστη μέχρι τη συσκευή δικτύωσης χρησιμοποιείται το Tunnel Mode. Αν θέλουμε περισσότερη προστασία μπορούμε να εφαρμόσουμε ένα ή περισσότερα σενάρια που υλοποιούνται στην περίπτωση 1.

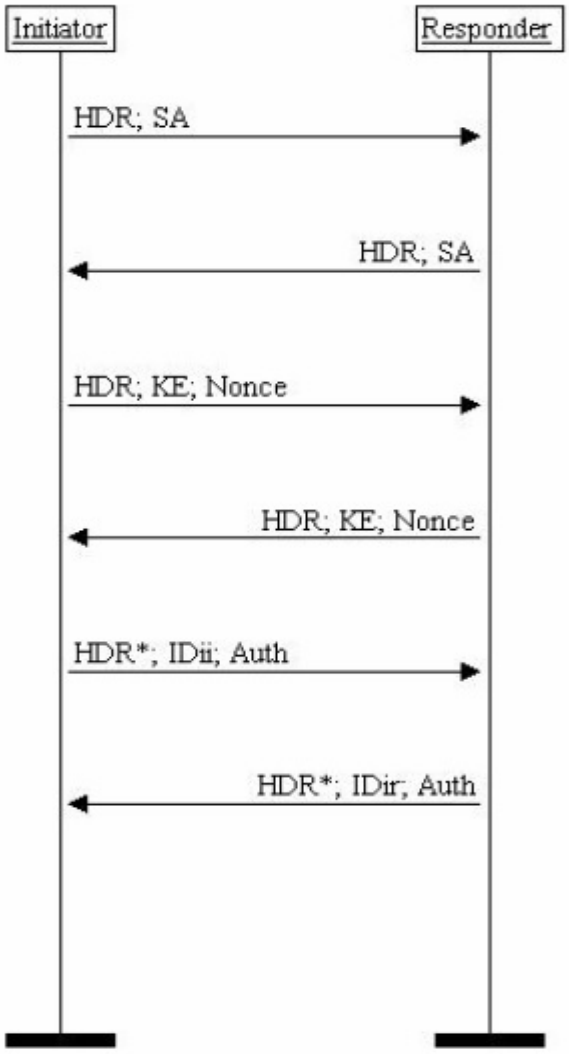



**Εικόνα 15.** Τέσσερις περιπτώσεις που μπορούν να εφαρμοστούν οι τρόποι λειτουργίας Transport και Tunnel (Stallings, W., 2010).

### 3.3.4 Το πρωτόκολλο Internet Key Exchange (IKE)

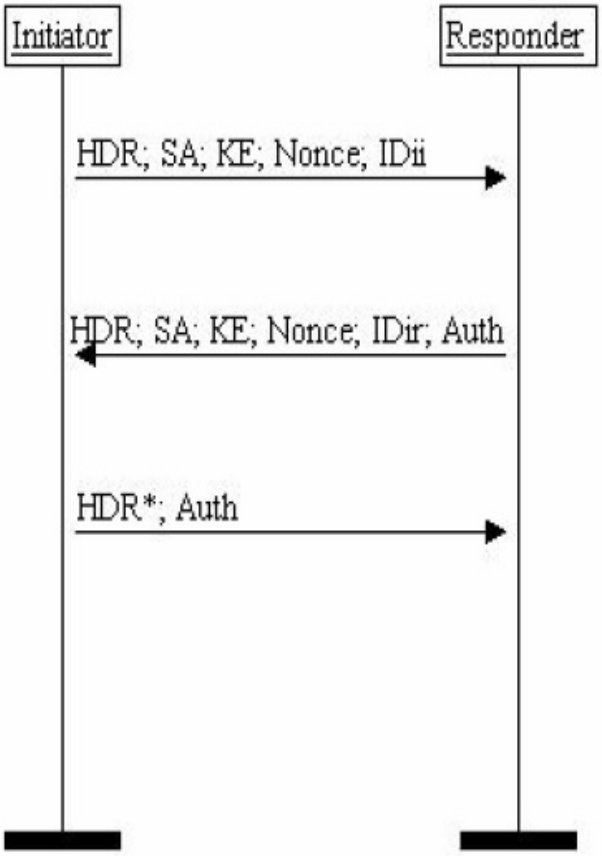
Το πρωτόκολλο Internet Key Exchange (IKE), χρησιμοποιείται για την διαπραγμάτευση των παραμέτρων ασφαλείας που θα χρησιμοποιηθούν σε μια σύνδεση, (αλγόριθμος αυθεντικοποίησης, αλγόριθμος κρυπτογράφησης, συναρτήσεις κατακερματισμού, μυστικά κλειδιά, και η διάρκεια ζωής των συσχετισμών ασφαλείας). Το IKE μπορεί να οριστεί χειροκίνητα, από το διαχειριστή του συστήματος , και αυτό ενδείκνυται μόνο στις περιπτώσεις μικρών δικτύων, γλυτώνοντας το σύστημα από το φόρτο του υπολογισμού συσχετισμούς ασφαλείας και αλγόριθμους ανταλλαγής κλειδιών, ή μπορεί να οριστεί αυτόματα με ένα πιο επιβαρυμένο λογισμικό.

Το IKE είναι ένας συνδυασμός των πρωτοκόλλων Internet Security Associations and Key Management (ISAKAMP), Oakley και Secure Key Exchange Mechanism (SKEME). Το ISAKAMP χρησιμοποιείται ως ένα πλαίσιο για την ανταλλαγή κλειδιών. Ο μηχανισμός ανταλλαγής κλειδιών προσφέρεται από το Oakley, το οποίο βασίζεται στο πρωτόκολλο ανταλλαγής κλειδιών του Diffie-Hellman, και επιπλέον προσφέρει αυθεντικοποίηση, Perfect Forward Security (PFS), και προστασία ταυτότητας. Το SKEME προσφέρει την ανωνυμότητα και τη γρήγορη ανανέωση του κλειδιού. Η καινούργια έκδοση του IKE το IKEv2 δεν υποστηρίζει τη χρήση των πρωτοκόλλων ISAKAMP και Oakley, παρόλα αυτά η διαλειτουργικότητα παραμένει η ίδια.




**Εικόνα 16.** Κύριος τρόπος ανταλλαγής κλειδιών (Μπόζης., Ε., Ζ., 2011).

Η φάση της διαπραγμάτευσης των παραμέτρων ασφαλείας μπορεί να γίνει με δυο τρόπους, το κύριο και τον επιθετικό. Στον κύριο τρόπο που φαίνεται στην εικόνα 16, παρατηρούμε ότι στην αρχή τα δυο πρώτα μηνύματα χρησιμοποιούνται για την εδραίωση της πολιτικής ασφαλείας. Στα επόμενα δυο μηνύματα ανταλλάσσονται κάποιες ποσότητες, που χρησιμοποιούνται για τον υπολογισμό των κλειδιών του Diffie-Hellman. Τέλος, τα δυο τελευταία μηνύματα είναι κρυπτογραφημένα με τα υπολογισμένα κλειδιά, και χρησιμοποιούνται για αυθεντικοποίηση, και επιπλέον για την ανταλλαγή κλειδιών, πιστοποιητικών και της ταυτότητας του χρήστη.




**Εικόνα17.** Ο επιθετικός τρόπος ανταλλαγής κλειδιών (Μπόζης., Ε., Ζ., 2011).

Ο επιθετικός τρόπος, που παρουσιάζεται στην εικόνα 17, είναι πολύ πιο ταχύς από τον κύριο τρόπο. Πραγματοποιείται σε μόλις τρία μηνύματα, συγχωνεύοντας τις πληροφορίες. Στο πρώτο μήνυμα, στέλνονται η πολιτική που θα χρησιμοποιηθεί, τιμές για τον υπολογισμό του κλειδιού Diffie-Hellman, τη ποσότητα Nonce καθώς και την ταυτότητα του χρήστη. Στο δεύτερο μήνυμα είναι σαν επιβεβαιωτικό του πρώτο μηνύματος, αφού ο χρήστης συμφωνεί με τις παραμέτρους που έχει λάβει και στέλνει τις δικές του τιμές για τον υπολογισμό του κλειδιού του Diffie-Hellman. Επιπλέον στέλνει την ταυτότητά του και ένα μήνυμα αυθεντικοποίησης. Όλα αυτά, δεν είναι κρυπτογραφημένα, με αποτέλεσμα η μέθοδος αυτή να είναι ιδιαίτερα επιρρεπής σε επιθέσεις. Στη συνέχεια τα δυο μέρη υπολογίζουν το κλειδί του Diffie-Hellman και ο πρώτος που ξεκίνησε την χειραψία, στέλνει ένα κρυπτογραφημένο μήνυμα που αυθεντικοποιεί τον εαυτό του (Kent S. , 2005α) (Kent S. , 2005β) (Orman, H., 1998) (Stallings, W., 2010, σσ. 270-300) (Stallings, W., 2014, σσ. 627-657) (Νούκας, Κ., Παππής, Χ., 2006, σσ. 49-52) (Κολοκοτώνης, 2014) (Λιμνίωτης, Κ., 2006, σσ. 14-30) (Μπόζης., Ε., Ζ., 2011, σσ. 18-35).

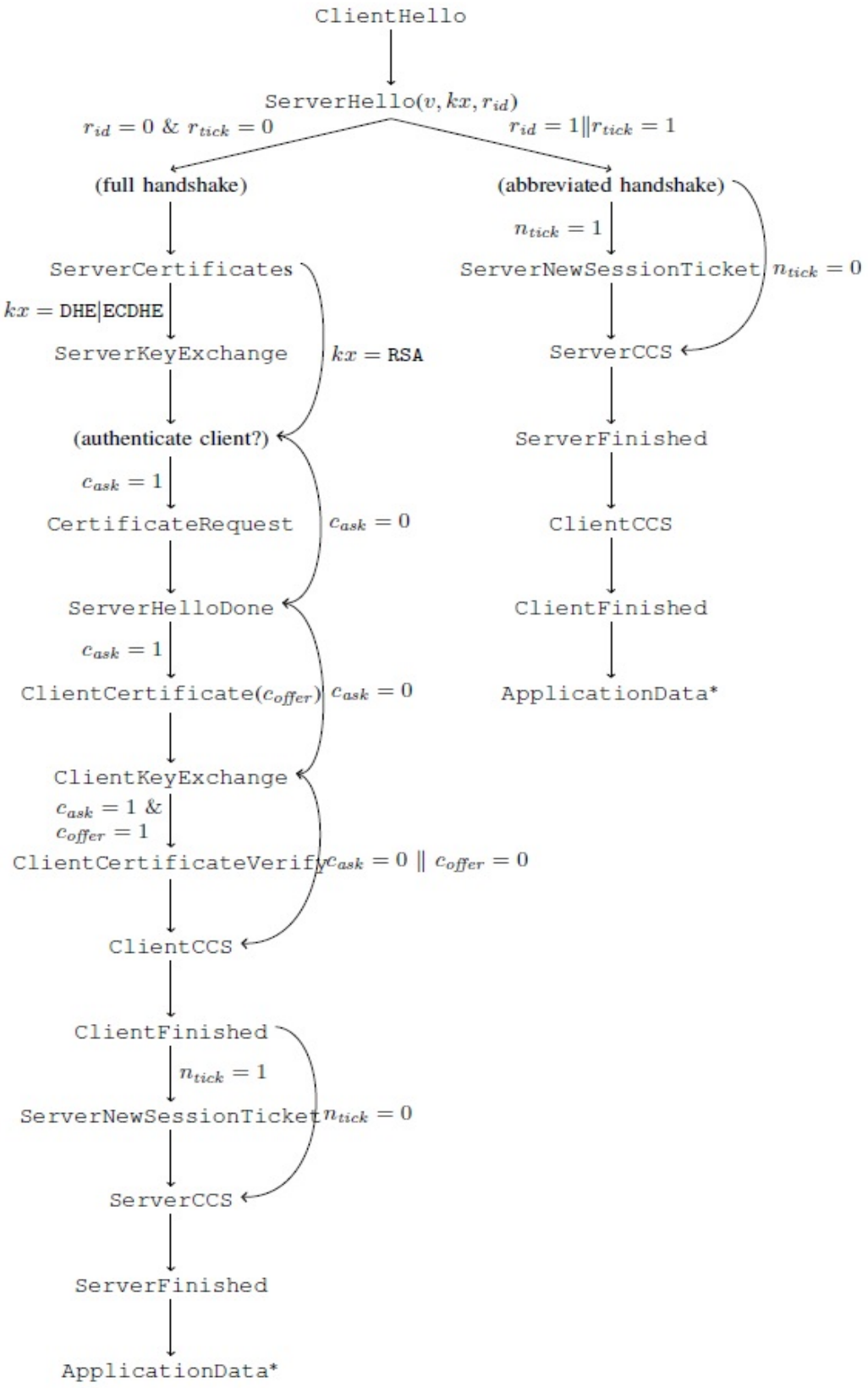
# Κεφάλαιο 4

## Επιθέσεις Ενδιάμεσου

Οι επιθέσεις ενδιαμέσου χρήστη (Man-In-The-Middle Attacks – MITM), αποτελούν ένα από τα διασημότερα είδη επιθέσεων στο διαδίκτυο. Δεν είναι ένα συγκεκριμένο είδος επίθεσης αλλά μια κατηγορία, όπου οι επιθέσεις έχουν ένα κοινό χαρακτηριστικό, ότι ο επιτιθέμενος εκμεταλλεύεται μια αδυναμία του συστήματος για να παρέμβει σε μια

συνομιλία στο διαδίκτυο. Παρόλο που συνδέσεις στο διαδίκτυο προστατεύονται από διάφορα πρωτόκολλα (SSL/TLS, IPsec, κ.α.), ο επιτιθέμενος εκμεταλλευόμενος τις ατέλειες του συστήματος παρουσιάζει το εαυτό του ως κάποιον άλλο με σκοπό την παρακολούθηση ή και την αλλοίωση των μηνυμάτων. Η προσπάθεια αυτή γίνεται κατά τη διάρκεια της χειραψίας και της ανταλλαγής κλειδιών. Έχουν πραγματοποιηθεί και αναλυθεί πάρα πολλές επιθέσεις, με διάφορες μεθόδους, όπως έκδοση ψεύτικων πιστοποιητικών, παραποίηση μηνυμάτων λόγω χρήσης αδύναμων συναρτήσεων κατακερματισμού κ.λπ. Οι πιο πρόσφατες επιθέσεις, όμως αφορούσαν τους αλγορίθμους ανταλλαγής κλειδιών και την αρχιτεκτονική του πρωτοκόλλου χειραψίας, οδήγησαν στην κατάρρευση του πρωτοκόλλου TLS v1.2 και την εξέταση για δημιουργία του TLS v1.3 (Asokan, N., Niemi, V., Nyberg, K., 2005) (Dacosta, I., Ahamad, M., Traynor, P., 2012) (Eberz, S., Strohmeier, M., Wilhelm, M., Martinovic, I., 2012) (Eriksson, 2003) (Nadhem, J., A., Kenneth, G., P., 2013).

Η συμφωνία των παραμέτρων ασφαλείας στο πρωτόκολλο TLS καλύπτεται από το πρωτόκολλο χειραψίας, όπως αναφέρθηκε στο προηγούμενο κεφάλαιο. Τα μηνύματα αμέσως μετά το ClientHello μήνυμα, διαδέχονται το ένα το άλλο σε μια αυστηρά τηρούμενη σειρά και μορφή. Παρόλα αυτά, η γενική μορφή του πρωτοκόλλου χειραψίας είναι ευέλικτη, επιτρέποντας την χρησιμοποίηση πιο παλιών εκδόσεων του TLS και διάφορους αλγορίθμους ανταλλαγής κλειδιών, που επιτρέπουν την παράληψη ορισμένων μηνυμάτων του πρωτοκόλλου χειραψίας, όπως βλέπουμε στην εικόνα 16.




**Εικόνα 18.** Το πρωτόκολλο χειραψίας για TLS για τις εκδόσεις 1.2, 1.1, 1.0 και για τους αλγορίθμους ανταλλαγής κλειδιών RSA, DHE, ECDHE (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015).

Όπως βλέπουμε στην εικόνα 18, υπάρχουν κάποιοι παράμετροι, που καθορίζουν την διαδοχή των μηνυμάτων στο πρωτόκολλο:

- **υ:** Η έκδοση του πρωτοκόλλου.
- **$k_x$:** Ο αλγόριθμος ανταλλαγής κλειδιών.
- **$r_{id}$:** Ο πελάτης ζητά την επανεκκίνηση της σύνδεσης που είχε με τον εξυπηρετητή, και ο εξυπηρετητής το αποδέχεται.
- **$r_{tick}$:** Ο πελάτης ζητά την επανεκκίνηση της σύνδεσης με διαφορετικό session_id που είχε με τον εξυπηρετητή, και ο εξυπηρετητής το αποδέχεται.
- **$c_{ask}$:** Ο εξυπηρετητής ζητά από τον πελάτη να αυθεντικοποιηθεί.
- **$c_{offer}$:** Ο πελάτης συμφωνεί στο να αυθεντικοποιηθεί.
- **$n_{tick}$:** Ο εξυπηρετητής στέλνει νέο session_id.

Αναλόγως των τιμών που παίρνουν οι παραπάνω παράμετροι καθορίζουν τη διαδοχή των μηνυμάτων στο πρωτόκολλο.

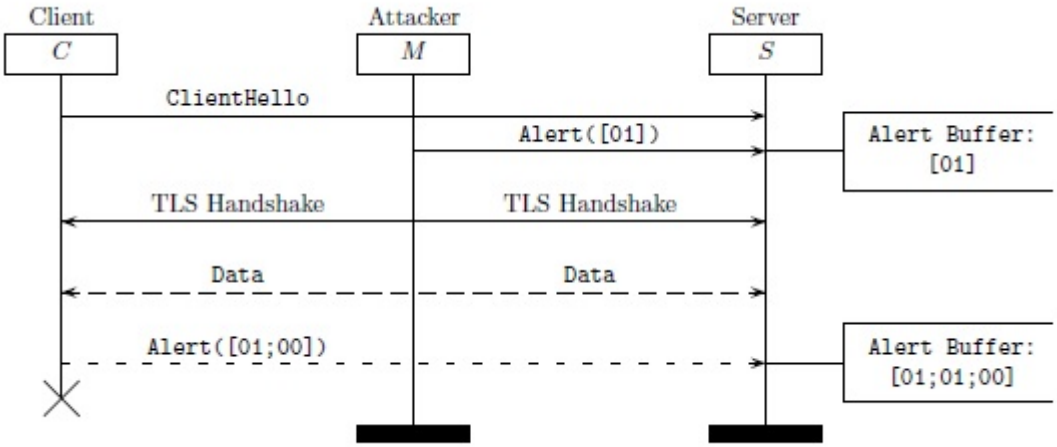
Για παράδειγμα οι τιμές που παίρνει το $r_{id}$ και το $r_{tick}$, καθορίζουν αν η χειραψία θα είναι πλήρη χειραψία (full handshake) ή σύντομη χειραψία (abbreviate handshake). Στη σύντομη χειραψία, τα περισσότερα μηνύματα παραλείπονται, και η σύνδεση πελάτη-εξυπηρετητή ενεργοποιείται ξανά βάση των παραμέτρων που είχαν συμφωνηθεί κατά την πλήρη χειραψία. Ακόμα ένα παράδειγμα είναι η παράμετρος $k_x$, που καθορίζει το αλγόριθμο ανταλλαγής κλειδιών. Αν ο αλγόριθμος είναι κάποιος ανώνυμος αλγόριθμος, όπως το Anonymous Diffie-Hellman, δεν προχωρά σε αυθεντικοποίηση του εαυτού του, παραλείποντας την αποστολή πιστοποιητικού. Αν ο αλγόριθμος χρειάζεται και αυθεντικοποίηση από μέρους του πελάτη, όπως το RSA with client Authentication, τότε το $c_{ask}$=1 και ζητείται το πιστοποιητικό του πελάτη μέσω του ClientCertificate. Αν ο πελάτης έχει κάποιο πιστοποιητικό το $c_{offer}$=1 και στέλνει το πιστοποιητικό μέσω του μηνύματος ClientCertificateVerify διαφορετικά $c_{offer}$=0 και παραβλέπεται το μήνυμα αυτό.

Όπως συμπεραίνεται, η συγχώνευση όλων των εκδόσεων του TLS, και όλων των αλγορίθμων ανταλλαγής κλειδιού, μπορεί να κρύβει παγίδες. Τα μηνύματα πρέπει να διαδέχονται το ένα το άλλο με μια αυστηρή σειρά. Για παράδειγμα το ServerCSS, που υποδηλώνει το τέλος και τη συμφωνία των παραμέτρων ασφαλείας, πρέπει να

ακολουθείται από το ServerFinished. Αν γίνει αποδεκτό πιο νωρίς ή πιο αργά μπορεί να οδηγήσει σε σοβαρά προβλήματα, όπως θα δούμε και παρακάτω. Εκτός αυτού, έστω και αν τα μηνύματα είναι στην ορθή σειρά, μπορεί να δημιουργήσουν προβλήματα, όπως η αποδοχή του ServerKeyExchange στον αλγόριθμο RSA ή η παράλειψή του στο EHDHE, μπορεί να έχει καταστροφικές συνέπειες. Επομένως η επανεξέταση της εισαγωγής επιπλέον μηνυμάτων πρέπει να εξεταστεί. Τέλος, ο διαχωρισμός της πλήρης από τη σύντομη χειραψίας μετά το ServerHello μπορεί να οδηγήσει σε λάθος επιλογή, μιας και ο εξυπηρετητής, στο επόμενο μήνυμα ξεκαθαρίζει τι είδους χειραψία θα ακολουθήσει.

## 4.1 Επίθεση Κατακερματισμένου Μηνύματος Προειδοποίησης (Fragmented Alert Attack)

Το πρωτόκολλο Alert, είναι ξεχωριστό πρωτόκολλο από αυτό της χειραψίας. Όσα μηνύματα ειδοποίησης σταλούν (alert messages) κατά τη διάρκεια του πρωτοκόλλου χειραψίας, δεν καταγράφονται στα Finished μηνύματα που στέλνονται από το πελάτη και τον εξυπηρετητή. Τα Finished μηνύματα περιλαμβάνουν, όλα τα μηνύματα που ανταλλάχθηκαν κατά την χειραψία για πιστοποίηση της ταυτότητας των δυο μερών. Επίσης τα μηνύματα ειδοποίησης μπορεί να σταλούν και κατά τη διάρκεια ανταλλαγής δεδομένων.

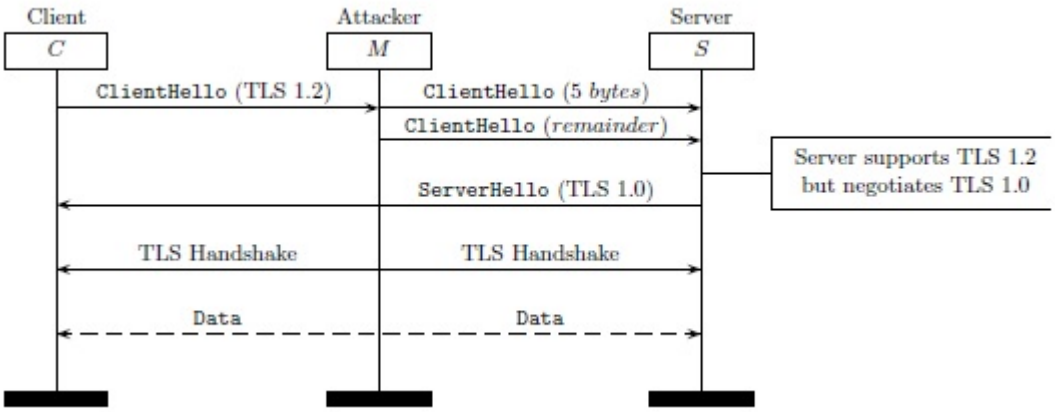



**Εικόνα 19.** Η επίθεση Alert (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015).

Ως γνωστό το μήνυμα ειδοποίησης, έχει μήκος δυο bytes. Κατά τη διάρκεια της επίθεσης ο επιτιθέμενος σπάζει το μήνυμα ειδοποίησης και κατά τη διάρκεια της χειραψίας στέλνει το ένα από τα δυο bytes [01], όπως φαίνεται και στην εικόνα 19. Ακολούθως ο εξυπηρετητής περιμένει να λάβει και το δεύτερο byte. Όταν το μήνυμα ειδοποίησης δημιουργείται από το πελάτη [01, 00] για σφάλμα στην αυθεντικοποίηση, ο εξυπηρετητής λαμβάνει [01, 01], με αποτέλεσμα να υποβαθμίζει το σφάλμα ειδοποίησης από κρίσιμο σε απλό προειδοποιητικό. Και έτσι αντί να τερματίσει την σύνδεση, τη συνεχίζει. Από τις έρευνες αναφέρεται ότι η εν λόγω επίθεση μπορεί να υλοποιηθεί σε openssl εξυπηρετητές (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015, σ. 5) (Bhargavan, K, Fournet, C., Kohlweiss, M., Pironti, A., Strub, P., Y., 2013, σ. 5) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

## 4.2 Επίθεση Κατακερματισμένου Client Hello Μηνύματος (Fragmented Client Hello Attack)

Η έκδοση του OpenSSL 1.0.1i, καθώς και προηγούμενες εκδόσεις, είναι επιρρεπείς σε ένα κενό ασφαλείας, που αναφέρεται στο Common Vulnerabilities and Exposures (CVE-2014-3511), και οδηγεί στην υποβάθμιση της έκδοσης του πρωτοκόλλου TLS στην έκδοση 1.0, ακόμα και αν πελάτης και εξυπηρετητής υποστηρίζουν πιο καινούργιες εκδόσεις αυτού.




**Εικόνα 20.** Επίθεση Fragmented ClientHello και υποβάθμιση του πρωτοκόλλου TLS v1.0 (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015).

Η επιτυχία της επίθεσης οφείλεται στον κατακερματισμό μηνυμάτων, που είναι επιτρεπτό στο πρωτόκολλο Record και δεν υπόκεινται σε οποιουσδήποτε περιορισμούς όσον αφορά το ελάχιστο μέγεθος των κατακερματισμένων μηνυμάτων, και δεν είναι ανιχνεύσιμο από το πρωτόκολλο Handshake. Έτσι ο επιτιθέμενος, μπορεί να κατακερματίσει το ClientHello μήνυμα και να επιτύχει υποβάθμιση του πρωτοκόλλου TLS στην έκδοση 1.0. Όπως φαίνεται και στην εικόνα 20, ο επιτιθέμενος κατακερματίζει το ClientHello σε δυο μηνύματα, ένα των 5 bytes και ένα δεύτερο που περιέχει τα υπόλοιπα bytes και τα στέλνει στον εξυπηρετητή (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015, p. 5) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

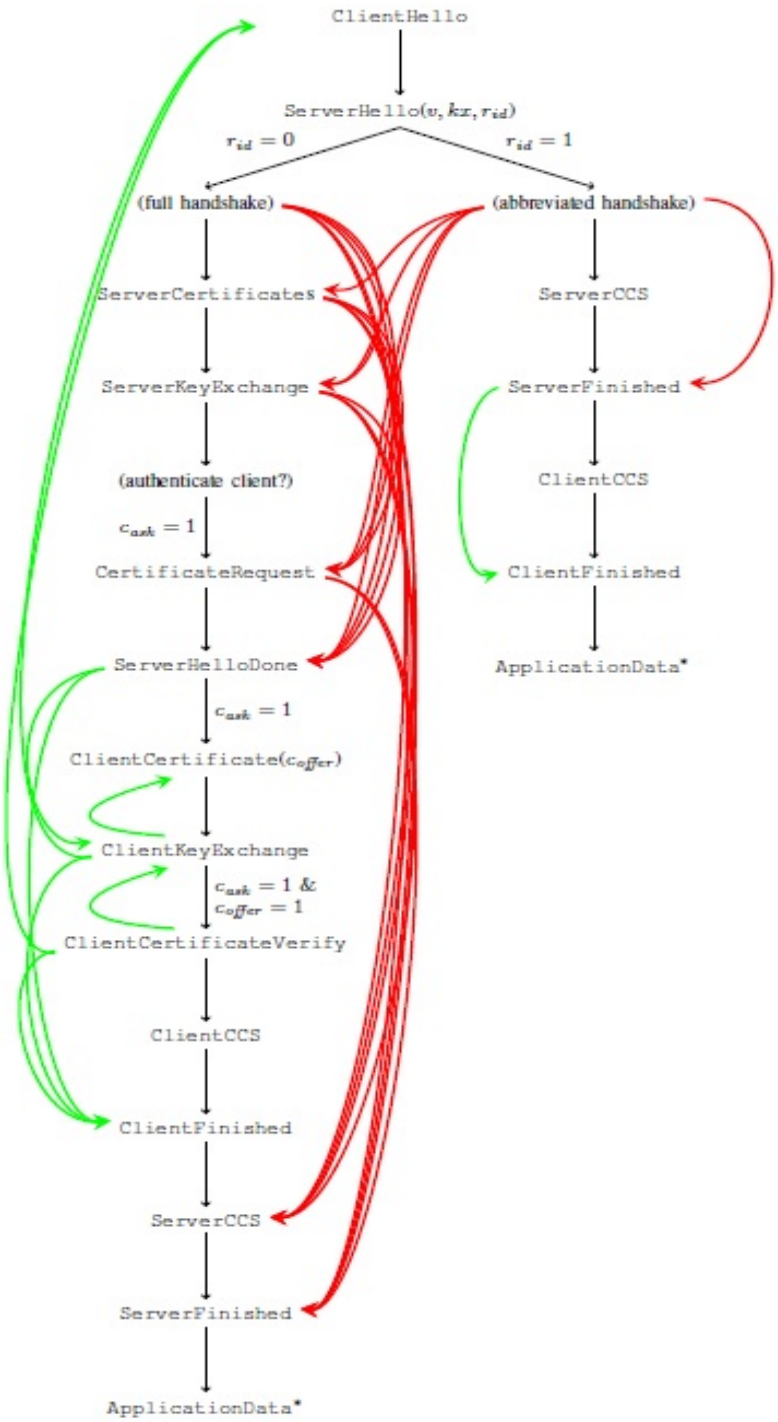
## 4.3 Επίθεση Παράληψης Μηνύματος (Skip Attack)

Η χρήση διαφορετικών αλγορίθμων για ανταλλαγή κλειδιού, οδηγεί στη διαφορετική ακολουθία μηνυμάτων κατά το πρωτόκολλο χειραψίας. Στην περίπτωση που ephemeral Diffie-Hellman επιλεχθεί, η αυθεντικοποίηση, επιτυγχάνεται μέσω του μηνύματος ServerKeyExchange, ενώ στην περίπτωση που επιλεχθεί ο RSA, το μήνυμα αυτό παρακάμπτεται, όπως βλέπουμε και στην εικόνα 18. Ως εκ τούτου κάποιες εφαρμογές του TLS επιτρέπουν την παράκαμψη μηνυμάτων, που θα έπρεπε να σταλούν, χωρίς κάποια ειδοποίηση κινδύνου ή σφάλματος, αφήνοντας έτσι τη σύνδεση πελάτη εξυπηρετητή εκτεθειμένη σε κινδύνους.

Στην εικόνα 21 η ροή των μηνυμάτων σε μια υλοποίηση του TLS σε ένα Java Secure Socket Extension (JSSE) εξυπηρετητή, καθώς και οι παρακαμπτήριες ροές. Με μαύρο χρώμα σημειώνεται η κανονική ροή των μηνυμάτων μαζί με τις νενομισμένες παρακάμψεις, όπως απεικονίζεται και στην εικόνα 18. Με πράσινο χρώμα είναι οι εσφαλμένες αποδεκτές παρακάμψεις από την πλευρά του εξυπηρετητή και με κόκκινο χρώμα φαίνονται οι εσφαλμένες αποδεκτά παρακάμψεις από το χρήστη.

Για να γίνει αντιληπτό, πως ένας κακόβουλος χρήστης Κ μπορεί να προσποιηθεί στο πελάτη Π ότι είναι ο εξυπηρετητής Ε παρουσιάζεται το εξής απλό σενάριο:

- Ο Π στέλνει ClientHello στον Ε.
- Ο Κ λαμβάνει το ClientHello και απαντά με ServerHello.
- Ο Κ στέλνει το πιστοποιητικό του Ε στο ServerCertificate.
- Ο Κ στέλνει ServerFinished, και η χειραψία τερματίζεται.




**Εικόνα 21.** Η υλοποίηση του TLS σε ένα JSSE εξυπηρετητή (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet,

C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015).

Στο τρίτο μήνυμα, ο κακόβουλος χρήσης στέλνει το πιστοποιητικό του εξυπηρετητή Ε. Παρόλο που το πιστοποιητικό είναι εσφαλμένο, είναι πολύ δύσκολο να γίνει αντιληπτό από τον πελάτη, ακόμα και μετά από μια προσεκτική εξέταση του, γιατί είναι πολύ εύκολο να βρεις ένα πλαστό πιστοποιητικό στο διαδίκτυο. Αμέσως μετά την αποστολή του πιστοποιητικού ο κακόβουλος χρήστης παρακάμπτει όλα τα επόμενα μηνύματα και τερματίζει τη χειραψία. Αυτό οδηγεί σε μη διαπραγμάτευση των παραμέτρων ασφαλείας μεταξύ πελάτη κακόβουλου χρήστη και τέλος στον υπολογισμό ενός κενού master_secret.

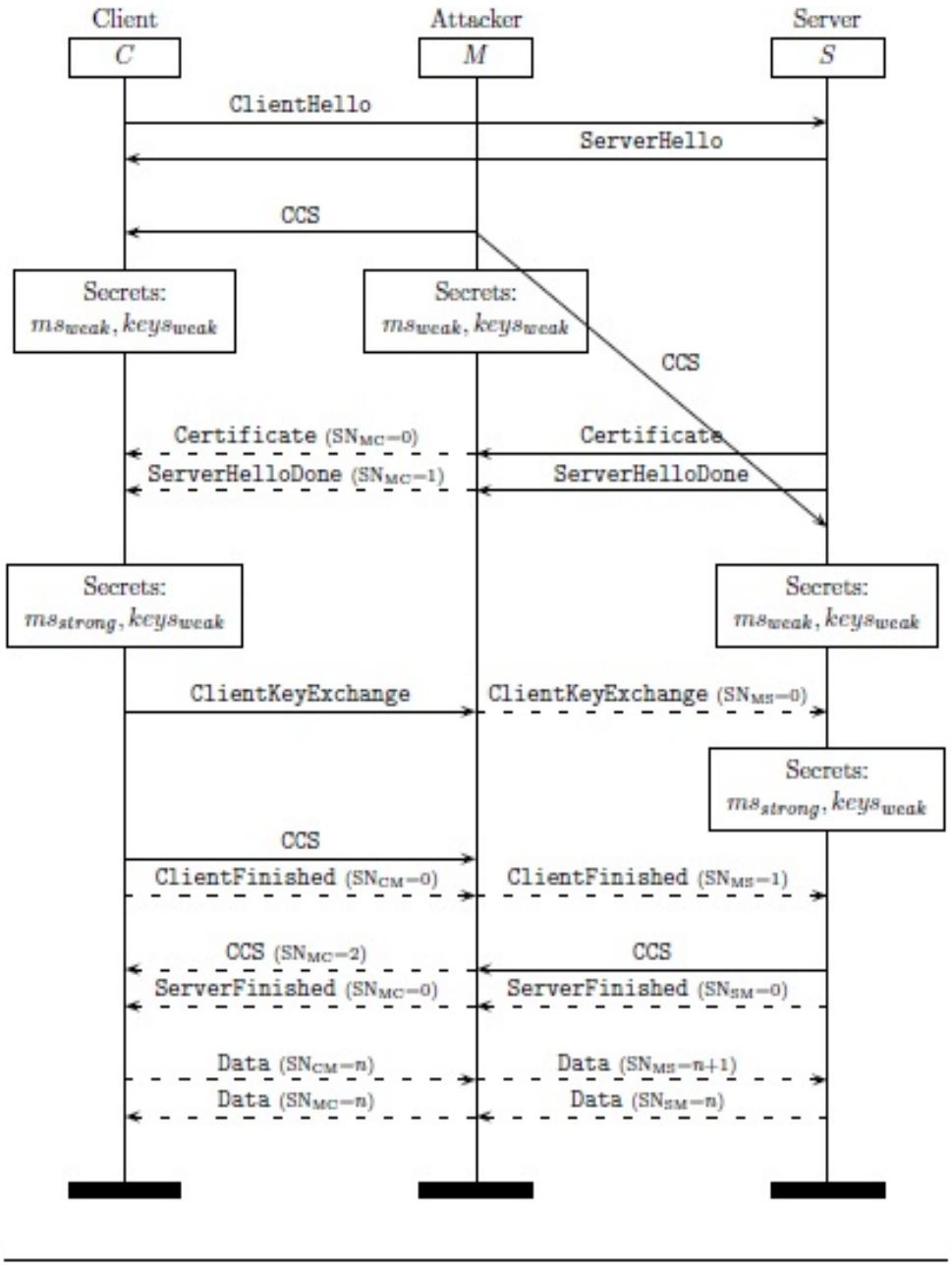
Επομένως, τα δεδομένα ανταλλάσσονται σε μορφή απλού κειμένου. Το μόνο που χρειάζεται να κάνει ο κακόβουλος χρήστης για να το πετύχει αυτό είναι να συμπεριλάβει στο κώδικα αυθεντικοποίηση μηνύματος που στέλνεται με το ServerFinished κομμάτια από τα τρία μηνύματα και ένα άδειο master_secret.

Κατά τον ίδιο τρόπο ένας κακόβουλος χρήστης, μπορεί να παρακάμψει όλα τα μηνύματα μέχρι το ServerFinished σε ένα CyaSSL εξυπηρετητή. Το μόνο που αλλάζει εδώ είναι ότι στο κώδικα αυθεντικοποίησης μηνύματος, ο κακόβουλος χρήστης πρέπει να προσθέσει ένα πίνακα που να περιέχει δώδεκα μηδενικά (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015, σσ. 9-11) (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015, σσ. 4-5) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

## 4.4 Επίθεση EarlyCCS

Η επίθεση αυτή βασίζεται στην παράκαμψη μηνυμάτων, όπως την προηγούμενη, αλλά και στην εσφαλμένη αποδοχή μηνυμάτων σε λάθος χρονικό σημείο. Η επιτυχία της επίθεσης, οφείλεται στο γεγονός ότι η βιβλιοθήκη OpenSSL αποδέχεται το μήνυμα ChangCipherSpec (CCS), το οποίο μπορεί να ανήκει στο πρωτόκολλο Change Cipher Spec, παρόλο που εμφανίζεται στο πρωτόκολλο Handshake, οποιανδήποτε στιγμή και αν αυτό σταλεί, όπως αναφέρεται στο Common Vulnerabilities and Exposures (CVE-

2014-0224) (Kikuchi, 2014). Επιπλέον δεν είναι ανιχνεύσιμο, γιατί ανήκει σε διαφορετικό πρωτόκολλο και δεν καταγράφεται στα Finished μηνύματα που ανταλλάσσονται στο τέλος της χειραψίας, όσες φορές και αν σταλεί.




**Εικόνα 22.** Η επίθεση EarlyCCS (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015).

Ο επιτιθέμενος στέλνει το CSS μήνυμα, μετά την αποστολή του ServerHello από τον εξυπηρετητή, προς τον πελάτη και τον εξυπηρετητή, όπως φαίνεται και στην εικόνα 22. Αφού ακόμα δεν έχει ολοκληρωθεί η διαπραγμάτευση των παραμέτρων ασφαλείας, η

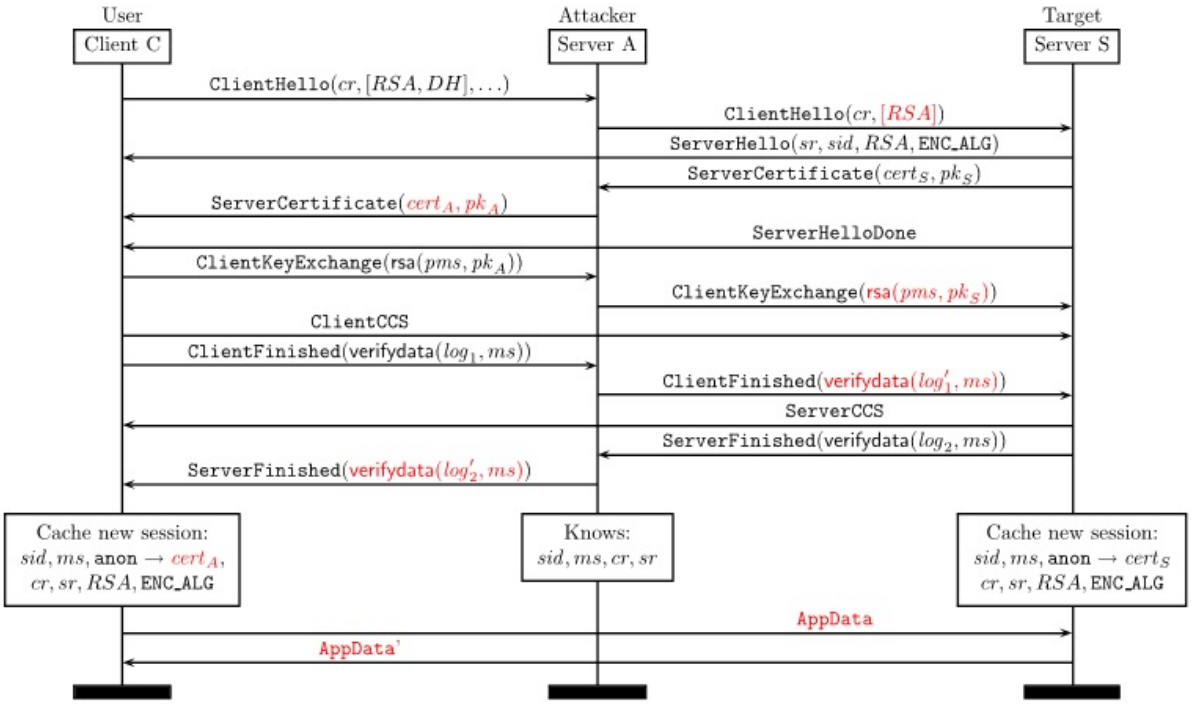
αποστολή του CSS την στιγμή αυτή οδηγεί τον πελάτη και εξυπηρετητή, στον υπολογισμό ενός 48-bytes μηδενικού master_secret. Επομένως τα κλειδιά που θα εξαχθούν βάση του master_secret, θα είναι αδύναμα, και ο επιτιθέμενος είναι πολύ εύκολο να τα υπολογίσει. Ακολούθως ο επιτιθέμενος αφήνει τα υπόλοιπα μηνύματα της χειραψίας να ανταλλαχθούν ελεύθερα μεταξύ εξυπηρετητή και πελάτη.

Στη συνέχεια, θα πραγματοποιηθεί ξανά ο υπολογισμός του master_secret με τις ανταλλασσόμενες ποσότητες, το οποίο θα είναι ισχυρό. Αυτό δεν επηρεάζει τον επιτιθέμενο, μιας και τα κλειδιά θα παραμένουν αδύναμα, όπως έχουν υπολογιστεί προηγουμένως. Το μόνο που χρειάζεται να κάνει είναι να σταματήσει τα μηνύματα ClientCCS και ServerCCS που στέλνονται από πελάτη και εξυπηρετητή, αντίστοιχα, και θα συνεχίσει να έχει τον έλεγχο της σύνδεσης, αποκρυπτογραφώντας τα δεδομένα με τα αδύναμα κλειδιά που υπολογίστηκαν παραπάνω. Μη αναχαίτηση του ClientCCS και ServerCCS, θα οδηγήσει σε εκ νέου υπολογισμό κλειδιών με τις παραμέτρους ασφαλείας που έχουν ανταλλαχθεί, και ο επιτιθέμενος δε θα είναι σε θέση να τα υπολογίσει, και η επίθεση θα αποτύχει (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015, σ. 7) (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015, σ. 6) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015) (Kikuchi, 2014).

## 4.5 Επίθεση Τριπλής Χειραψίας (Triple Handshake Attack)

Η επίθεση της Τριπλής Χειραψίας (Triple Handshake) μπορεί να θυμίζει τις επιθέσεις που πραγματοποιήθηκαν το 2009, κατά την επαναδιαπραγμάτευση της σύνδεσης με τον εξυπηρετητή, και περιγράφονται στα (Ray, M., Dispensa, S., 2009) και (Rex, M., 2009) και διορθώθηκαν στο RFC 5746 (Rescorla, E., Ray, M., Dispensa, S., Oskov., N., 2010). Η διαφορά της Triple Handshake από τις προηγούμενες επιθέσεις είναι ότι, ενώ οι προηγούμενες επιθέσεις βασιζόντουσαν στην αδυναμία τις κρυπτογραφικής δέσμευσης, η Triple Handshake βασίζεται στην αδυναμία της δέσμευσης διασταυρούμενων συνδέσεων. Επιπλέον, για την Triple Handshake, δε χρειάζεται ένα ενεργό επιτιθέμενο, αλλά ένα κακόβουλο εξυπηρετητή.

Αν υποθέσουμε ότι ο πελάτης C θέλει να συνδεθεί στον εξυπηρετητή S, αλλά συνδέεται στον κακόβουλο εξυπηρετητή A. Ο A στη συνέχεια συνδέεται ως C' στον S. Ο A στην ουσία αναλαμβάνει ένα ρόλο εξυπηρετητή μεσολάβησης (proxy server), προωθώντας τα μηνύματα από τη σύνδεση C-A στη A-S. Για να μην γίνει αντιληπτός ο A θα πρέπει την στιγμή που S στέλνει το πιστοποιητικό του, ο A να το ανακόψει και να στείλει το δικό του. Παρόλα αυτά, αν ο πελάτης είναι καχύποπτος και εξετάσει το πιστοποιητικό, μπορεί να ανακαλύψει ότι το πιστοποιητικό δεν ανήκει στον S. Για να αποκλείσει ένα τέτοιο ενδεχόμενο, ο A θα πρέπει να κατέχει το ιδιωτικό κλειδί του S, πράγμα σχεδόν αδύνατο.




**Εικόνα 23.** Επίθεση Τριπλής Χειραψίας, με εξ ανάγκαση χρησιμοποίησης του αλγορίθμου ανταλλαγής κλειδιού RSA (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

Οπόταν υποθέτουμε ότι ο C δεν εξετάζει το πιστοποιητικό του A, και έτσι ξεγελιέται, και πιστεύει ότι είναι συνδεδεμένος στον S. Η επιτυχία της επίθεσης οφείλεται στην ικανότητα του να εξαναγκάσει και τις δυο συνδέσεις να χρησιμοποιήσουν τα ίδια κλειδιά. Αφού το καταφέρει αυτό θα μεταβληθεί σε ένα παθητικό ωτακουστή, και δύσκολα θα μπορεί να ανιχνευτεί. Στην κρυπτογραφική βιβλιογραφία αυτή η επίθεση είναι γνωστή ως αγνώστου διαμοιραζόμενου κλειδιού επίθεση (unknown key-share

attack), και είναι επιτυχείς όταν δυο μέρη ανταλλάσσουν τα κλειδιά τους χωρίς να ξέρουν στην ουσία, ποιο είναι το άλλο μέρος. Η επίθεση μπορεί να πετύχει σε τρεις διαφορετικές περιπτώσεις.

Στην πρώτη περίπτωση, που απεικονίζεται στην εικόνα 23 και περιγράφεται παρακάτω, ο Α λαμβάνει το ClientHello μήνυμα από τον C. Αν στους προτεινόμενους αλγορίθμους ανταλλαγής κλειδιών, περιλαμβάνονται και άλλοι εκτός από τον RSA, τότε τους αφαιρεί και προωθεί το τροποποιημένο μήνυμα στον S. Ο S απαντά με το ServerHello και συμφωνεί στο αλγόριθμο RSA, μη έχοντας άλλη επιλογή. Στη συνέχεια στέλνει το πιστοποιητικό του μαζί με το δημόσιο του κλειδί. Ο Α τροποποιεί το μήνυμα, τοποθετώντας το δικό του πιστοποιητικό και το δικό του δημόσιο κλειδί. Όλοι, τώρα γνωρίζουν τις παραμέτρους $c_r$, $s_r$ και sid που ανταλλάσσονται στα Hello μηνύματα. Το μόνο που απομένει είναι το master_secret (ms). Ο C υπολογίζει το pre_master_secret (pms), το κρυπτογραφεί με το δημόσιο κλειδί του Α και το στέλνει στον Α. Ο Α το αποκρυπτογραφεί με το ιδιωτικό του κλειδί και το κρυπτογραφεί με το δημόσιο κλειδί του S, και το στέλνει σε αυτόν. Τώρα και οι δυο συνδέσεις έχουν το ίδιο ms, και συνεπώς και τα ίδια κλειδιά. Το μόνο που απομένει είναι η τροποποίηση των Finished μηνυμάτων, λόγω του ότι ανταλλάχτηκαν διαφορετικά πιστοποιητικά σε κάθε σύνδεση.

Στην περίπτωση που ο C δεν υποστηρίζει τον αλγόριθμο ανταλλαγής κλειδιών RSA, αλλά υποστηρίζει τον ephemeral Diffie-Hellman (DHE), η επίθεση μπορεί και πάλι να πραγματοποιηθεί. Όπως φαίνεται και στην εικόνα 24, ο Α κατά τον ίδιο τρόπο με προηγουμένως επεμβαίνει, αλλάζοντας το ClientHello μήνυμα ώστε ο S να έχει ως μόνη επιλογή τον DHE. Στη συνέχεια αλλάζει τα πιστοποιητικά, στέλνοντας τα δικά του. Επιπλέον επεμβαίνει στο ServerKeyExchange μήνυμα, που περιέχει τον αριθμό $P_s$, και περιγράφεται από τη σχέση 4.1.

$$P_S = g^{K_s} \cdot \mathrm{mod}(p)$$

(4.1)

Ακολούθως, αντικαθιστά τον πρώτο αριθμό p με ένα περιττό αριθμό, που δεν είναι πρώτος. Ο αριθμός πρέπει να είναι πολύ προσεκτικά επιλεγμένος, ώστε το ms να είναι ίσο με το $P_S$ και περιγράφεται από τη σχέση 4.2.

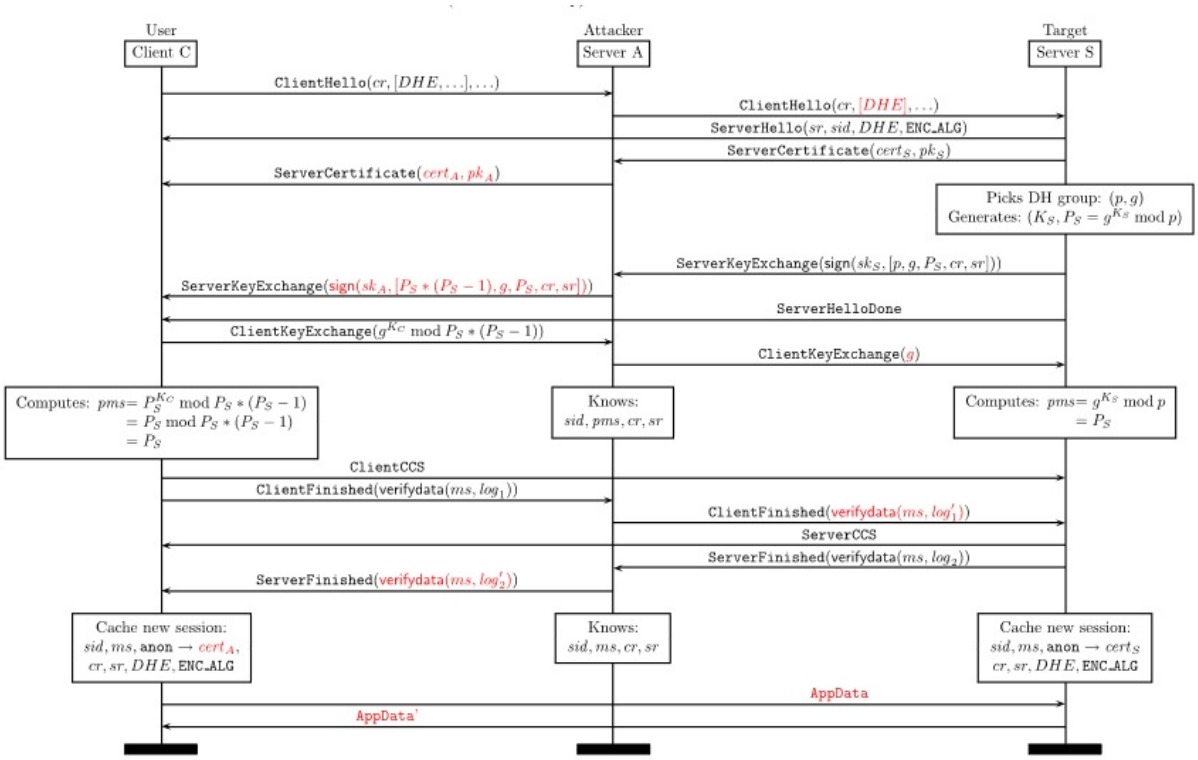
$$p' = P_S \cdot (P_S - 1)$$

(4.2)

Ο C υπολογίζει το $P_C$, που περιγράφεται από τη 4.3, και το στέλνει στον A.

$$P_C = g^{K_C} \bmod (p')$$

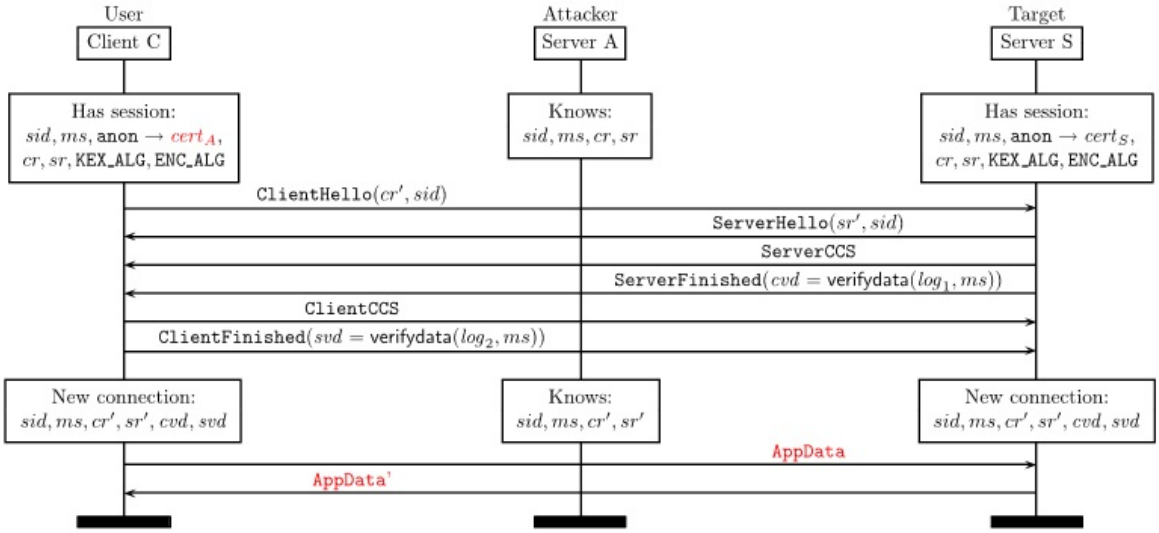
(4.3)




**Εικόνα 24.** Επίθεση Τριπλής Χειραψίας με εξ ανάγκαση χρησιμοποίησης του αλγορίθμου ανταλλαγής κλειδιού DHE (BHARGAVAN, K., DELIGNAT-LAVAUD, A., FOURNET, C., KOHLWEISS, M., ZANELLA-BÉGUELIN, S., ZINZINDOHOUÉ, J., K., BEURDOUCHE, B., KOBEISSI, N., PIRONTI, A., STRUB, P., Y., 2015).

Ο A το αντικαθιστά με την τιμή $P_C$ με το g και το στέλνει στον S. Τώρα και οι δύο συνδέσεις έχουν τα ίδια pms και ms. Το μόνο που απομένει είναι η αλλαγή στα Finished μηνύματα, όπως είχαμε και στην προηγούμενη περίπτωση.

Η τελευταία περίπτωση που μπορεί να επιτύχει η επίθεση είναι αυτή της επαναδιαπραγμάτευσης μιας εκ των δυο πιο πάνω περιπτώσεων, μέσω της σύντομης χειραψίας (abbreviate handshake), όπως φαίνεται και στην εικόνα 25. Εφόσον οι παραμέτροι, μένουν οι ίδιοι, ο A συνεχίζει να έχει τον έλεγχο της σύνδεσης και επομένως το μόνο που έχει να κάνει είναι η προώθηση των πακέτων από τον C στον S, και αντίθετα.




**Εικόνα 25.** Επίθεση Τριπλής Χειραψίας σε σύντομη χειραψία (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

Η μόνη περίπτωση για να αποτύχει η επίθεση στην περίπτωση της σύντομης χειραψίας, είναι κατά την επαναδιαπραγμάτευση, ο S να ζητήσει αυθεντικοποίηση του C. Τότε η διαδικασία της διαπραγμάτευσης ξεκινά από την αρχή. Ο A δεν είναι σε θέση να προσποιείται το C και έτσι τα μηνύματα από τον S φτάνουν κατευθείαν στο C χωρίς να μπορούν να τροποποιηθούν. Επομένως ο C παραλαμβάνει το πιστοποιητικό του S αυτή τη φορά, και απαντά με το δικό του πιστοποιητικό. Ακολούθως υπολογίζουν ένα καινούργια ms, το οποίο ο A αγνοεί. Κάπου εδώ χάνει και την σύνδεση, μιας και τα δεδομένα είναι κρυπτογραφημένα με νέα κλειδιά που ο ίδιος δε γνωρίζει.

Το περίεργο είναι η αντίδραση του C στην παραλαβή του νέου πιστοποιητικού από τον S, αφού προηγουμένως ήταν κατείχε το πιστοποιητικό του A. Ο C δεν αντιδρά καθόλου στην αλλαγή αυτή, τερματίζοντας την σύνδεση, ούτε καν να στείλει κάποιο προειδοποιητικό μήνυμα. Αυτό οφείλεται στο γεγονός ότι το TLS επιτρέπει τις αλλαγές αυτές στα πιστοποιητικά, θεωρώντας, ότι σε μια μακρόχρονη σύνδεση το πιστοποιητικό μπορεί να έχει αποσυρθεί, ή μπορεί να θεωρείται αδύναμο και ευπαθή και επομένως ευνοεί την αντικατάσταση τους αδιαμαρτύρητα (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015, σ. 5) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., Strub, P., Y., 2014, σσ. 7-9).

## 4.6 Επίθεση FREAK

Το 1999, μέσα σε μερικές ώρες στέφθηκε με επιτυχία το σπάσιμο του αλγόριθμου RSA με μήκος κλειδιού που δεν ξεπερνά τα 512-bits. Άρα με τους υπολογιστές που έχουμε σήμερα, ο αλγόριθμος με το αυτό το μήκος κλειδιού, μπορεί να θεωρείται ήδη σπασμένος. Το 2000, στα πρωτόκολλα SSL v3.0 και TLS v1.0, έγινε εξαγωγή παρόμοιων αλγορίθμων που παρουσίαζαν την ίδια ευπάθεια, από την κυβέρνηση των Ηνωμένων Πολιτειών Αμερικής (ΗΠΑ). Στην έκδοση του TLS v1.1 οι αλγόριθμοι αυτοί απομακρύνθηκαν εντελώς. Παρόλα αυτά, μερικές βιβλιοθήκες που υλοποιούν το TLS, όπως η OpenSSL, συνέχιζαν να υποστηρίζουν σιωπηρά αυτούς τους εξαγόμενους (EXPORT) αλγορίθμους , με την αιτιολογία του ότι μερικοί εξυπηρετητές και πελάτες συνέχιζαν να τους υποστηρίζουν. Όπως αποκαλύφθηκε αργότερα, η κυβέρνηση των ΗΠΑ πίεζε για τη συνέχιση της υποστήριξης τέτοιων αλγορίθμων για να μπορεί να αποκρυπτογραφεί και να παρακολουθεί όλη την ξένη κρυπτογραφία.

Η επίθεση πραγματοποιείται με την αντικατάσταση του προτεινόμενου, από τον πελάτη C, αλγορίθμου RSA με τον RSA_EXPORT, από τον επιτιθέμενο A και ονομάστηκε FREAK (Factoring RSA Export Keys). Η επίθεση φαίνεται στα παρακάτω βήματα:

- Ο C στέλνει ClientHello με αλγόριθμο ανταλλαγής κλειδιών RSA ως επιλογή.
- Ο A αντικαθιστά τον RSA με τον RSA_EXPORT και στέλνει το μήνυμα στον S.
- Ο S στέλνει το ServerHello με το RSA_EXPORT επιλεγμένο.
- Ο A αντικαθιστά το RSA_EXPORT με το RSA
- Ο S στέλνει το πιστοποιητικό του υπογεγραμμένο από ένα ισχυρό κλειδί.
- Ο S στέλνει το ServerKeyExchange που περιέχει ένα αδύναμο δημόσιο κλειδί ephemeral RSA (modulus N) μήκους 512-bit.
- Ο C στέλνει ClientKeyExchange, ClientCCS και ClientFinished.
- Ο A παραγοντοποιεί το N, και βρίσκει το ιδιωτικό κλειδί. Άρα μπορεί να αποκρυπτογραφήσει το pre_master_secret και να βρει το master_secret.
- Ο A στέλνει το ServerCCS και ServerFinished και μπορεί να διαβάσει τα δεδομένα.

Απ' εδώ και μπρος ο επιτιθέμενος Α μπορεί να διαβάζει τα μηνύματα. Μοναδική πρόκληση στην όλη υπόθεση για τον επιτιθέμενο είναι η παραγοντοποίηση του N. Δεδομένου ότι ο υπολογισμός, συνεχώς, καινούργιου RSA ιδιωτικού κλειδιού είναι λιγάκι ασύμφορος, μπορούμε να θεωρήσουμε ότι ένας εξυπηρετητής, θα χρησιμοποιεί για κάποιες μέρες το ίδιο κλειδί. Επομένως, δεν είναι ανάγκη ο υπολογισμός του ιδιωτικού κλειδιού να πραγματοποιηθεί την ώρα της σύνδεσης, αφού θα χρειαστούν κάποιες ώρες γι' αυτό, αλλά μπορεί να γίνουν πρώτα οι υπολογισμοί για το κλειδί και ακολούθως να πραγματοποιηθεί η επίθεση. Εκφράζεται η πεποίθηση ότι με 100 δολάρια στις υπηρεσίες νεφέλης του Amazon EC2, το κλειδί μπορεί να βρεθεί μέσα σε 12 ώρες (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015, σσ. 11-12) (Bhargavan, K, Fournet, C., Kohlweiss, M., Pironti, A., Strub, P., Y., 2013, σ. 7) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

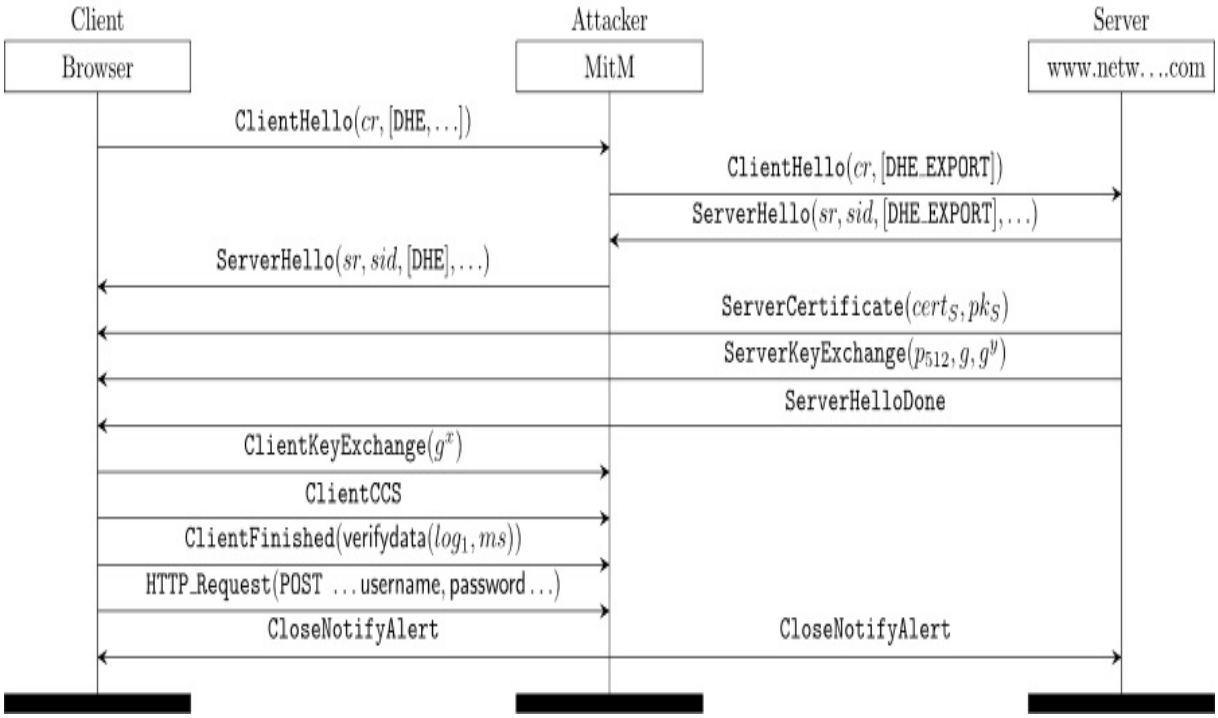
## 4.7 Επίθεση Logjam

Η επίθεση Logjam θυμίζει την προηγούμενη επίθεση FREAK, με τη διαφορά ότι πραγματοποιείται στον αλγόριθμο ανταλλαγής κλειδιού ephemeral Diffie-Hellman (DHE) με μικρό μήκος κλειδιού. Οι αλγόριθμοι αυτοί, όπως και αυτοί του RSA που περιεγράφηκαν πιο πάνω, εξάχθηκαν το 2000 από την κυβέρνηση των ΗΠΑ και περιγράφονται ως DHE_EXPORT. Πολλοί εξυπηρετητές συνέχιζαν την υποστήριξη ενός κυρίου αλγορίθμου DHE με κλειδί μήκους 1024-bit και άνω, και ενός DHE_EXPORT. Ο λόγος της συνέχισης της υποστήριξης του αδύναμου DHE_EXPORT, είναι ότι οι φυλλομετρητές (browsers), δεν είχαν το DHE_EXPORT στους υπό διαπραγμάτευση αλγορίθμους, και έτσι οι συνδέσεις θεωρούνταν ασφαλείς.

Σύμφωνα με την κρυπτανάλυση που έγινε και δημοσιεύθηκε στο (Adrian, D., Bhargavan, K., Durumeric, Z., Caudry, P., Green, M., Halderman, A., J., Heninger, N., Springall, D., Thome, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanela-Beguelin, S., Zimmerman, P., 2015), ο Apache εξυπηρετητής στις εκδόσεις του από 2.1.5 μέχρι 2.4.7 είχε προ εμφυτευμένο να χρησιμοποιά ως προκαθορισμένο το DHE_EXPORT. Περίπου το 7% του ενός εκατομμυρίου δημοφιλέστερων ιστοσελίδων χρησιμοποιούσαν το DHE_EXPORT. Ακόμα και η βιβλιοθήκη OpenSSL χρησιμοποιούσε το DHE_EXPORT. Με τη χρήση του

εργαλείου CADO-NFS έγινε μια προεργασία στους πρώτους αριθμούς των δυο πιο πάνω περιπτώσεων, η οποία κράτησε δυο εβδομάδες. Η διαδικασία υποβοήθησε στην επίλυση του προβλήματος του διακριτού λογαρίθμου, που επιφέρει ο Diffie-Hellman, σε μερικά λεπτά. Λαμβάνοντας δε υπ' όψη ότι πολλοί εξυπηρετητές Apache χρησιμοποιούν τους ίδιους πρώτους αριθμούς, η επίθεση μπορεί να εκταθεί και σ' αυτούς. Η επίθεση Logjam πραγματοποιήθηκε με τέσσερις παραλλαγές.

Η πρώτη παραλλαγή πραγματοποιήθηκε σε εξυπηρετητές που υποστήριζαν DHE με 512-bit κλειδί. Η επιτυχία της επίθεσης αυτής οφείλεται στο γεγονός ότι το αλγόριθμος ανταλλαγής κλειδιού χρησιμοποιούσε ένα αδύναμο κλειδί, το οποίο μπορούσε να σπάσει με μια προεργασία μιας εβδομάδας, και να χρησιμοποιηθεί αργότερα στην επίθεση. Ένας από τους ισοτόπους που παρουσίαζαν μια τέτοια ευπάθεια είναι ο ισότοπος του FBI https://tips.fbi.gov/, τον Απρίλιο του 2015. Αργότερα το Μάιο του ιδίου έτους, αυτό διορθώθηκε, αφαιρώντας τον αδύναμο αυτό αλγόριθμο από τις επιλογές του εξυπηρετητή που φιλοξενούσε τον ισότοπο.




**Εικόνα 26.** Επίθεση Logjam με υποβιβασμό του αλγορίθμου ανταλλαγής κλειδιών σε DHE_EXPORT και υποστήριξη του πελάτη TLS False Start για μετέπειτα αποκρυπτογράφηση των δεδομένων (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015)

Η δεύτερη παραλλαγή αφορούσε την υποβάθμιση του αλγορίθμου ανταλλαγής κλειδιών από DHE σε DHE_EXPORT από τον επιτιθέμενο και το μετέπειτα σπάσιμο του κλειδιού. Απαραίτητη προϋπόθεση για την επιτυχία της επίθεσης αυτής είναι, εκτός από την υποβάθμιση του αλγορίθμου, ο φυλομετρητής του πελάτη να υποστηρίζει το TLS False Start (LANGLEY, A., MODADUGU, N., MOELLER, B., 2010). Στην περίπτωση που ο φυλομετρητής υποστήριζε το TLS False Start, ξεκινούσε τον διαμοιρασμό δεδομένων με τον εξυπηρετητή, χωρίς να περιμένει τα μηνύματα επιβεβαίωσης απ' αυτόν ServerCCS και ServerFinished. Την ευπάθεια αυτή παρουσιάζουν ο Firefox 35 και ο Google Chrome 41, καθώς και προηγούμενες τους εκδόσεις.

Η εικόνα 26 παρουσιάζει την επίθεση την στιγμή της χειραψίας.

- Ο C στέλνει το ClientHello μήνυμα με ένα από τους αλγόριθμους ανταλλαγής κλειδιών τον DHE.
- Ο A παραποιεί το μήνυμα, αφήνοντας το DHE_EXPORT ως μόνη επιλογή αλγορίθμου ανταλλαγής κλειδιών και το προωθεί στον S.
- Ο S απαντά με το ServerHello και αποδεχόμενος το DHE_EXPORT.
- Ο A τροποποιεί το μήνυμα, τοποθετώντας το DHE στη θέση του DHE_EXPORT και προωθεί το μήνυμα στον C.
- Ο S στέλνει το πιστοποιητικό του υπογεγραμμένο από ένα δυνατό ιδιωτικό κλειδί.
- Ο S στέλνει το ServerKeyExchange μήνυμα με τις παραμέτρους για το 512-bit κλειδί, και το ServerHelloDone.
- Ο C στέλνει το ClientKeyExchange και στέλνει την επιβεβαίωση του μέσω του ClientCCS και ClientFinished, περιμένοντας την επιβεβαίωση από το εξυπηρετητή.
- Ο C ξεκινά να στέλνει δεδομένα στον εξυπηρετητή, χωρίς να λάβει την επιβεβαίωση από τον εξυπηρετητή. Ο A τερματίζει την σύνδεση πελάτη εξυπηρετητή. Αν ο C ζητήσει επανεκκίνηση της σύνδεσης, ο A την αφήνει να συνεχιστεί απρόσκοπτα.

Παρατηρούμε ότι παρόλο που ο εξυπηρετητής στέλνει ένα πιστοποιητικό, υπογεγραμμένο με ένα ισχυρό κλειδί, ο πελάτης αποδέχεται τις αδύναμες παραμέτρους

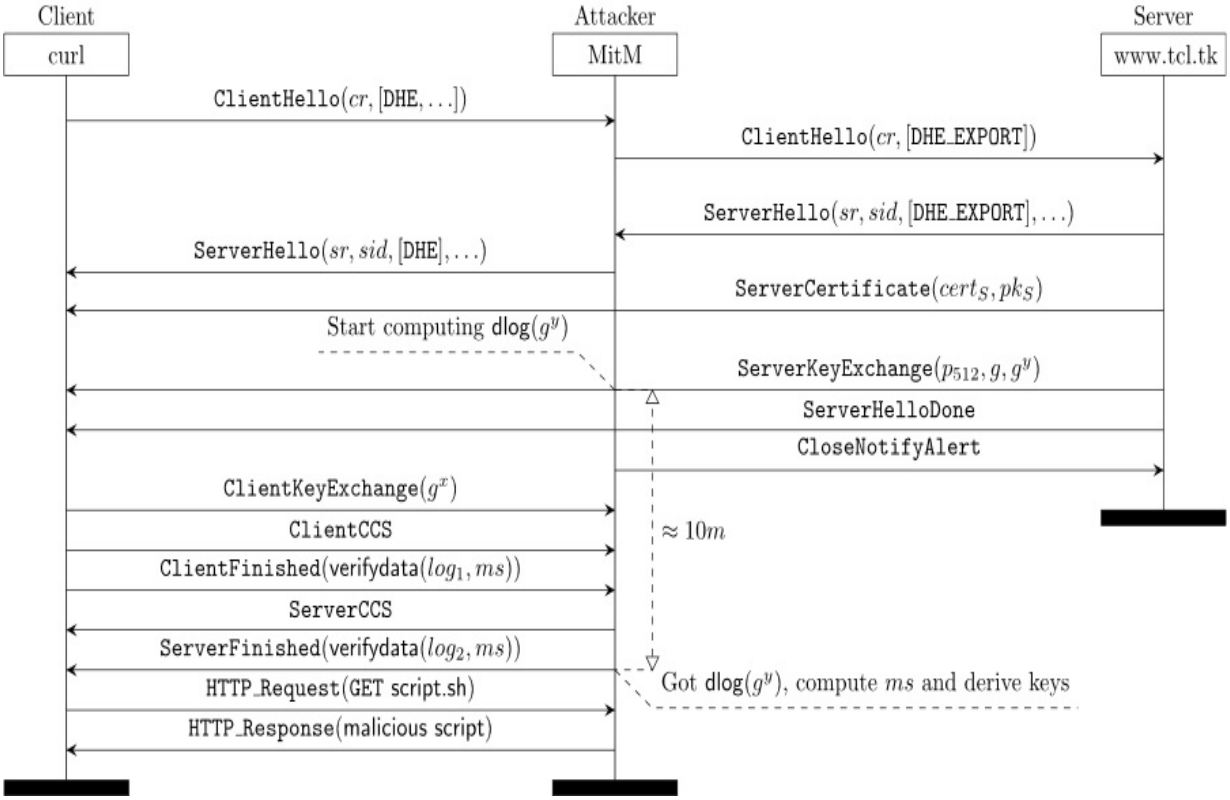
που προσφέρονται από το εξυπηρετητή στο επόμενο μήνυμα, θεωρώντας τις ορθές, αφού προσφέρονται από τον εξυπηρετητή. Ενώ στη συνέχεια στέλνει σημαντικά δεδομένα όπως είναι το username και το password. Ο επιτιθέμενος σταματά τη σύνδεση και ξεκινά να υπολογίζει το κλειδί. Αυτό θα είναι εφικτό σε μια εβδομάδα, πράγμα που δεν τον επηρεάζει αφού η σύνδεση δεν χρειάζεται να είναι πια ενεργή και τα δεδομένα βρίσκονται στα χέρια του.

Η επόμενη περίπτωση μοιάζει πολύ με την προηγούμενη, μόνο που η διάσπαση του κλειδιού γίνεται κατά την σύνδεση. Λόγω αυτού, απαιτείται να γίνει μια προεργασία προηγουμένως, ούτως ώστε η διάσπαση του κλειδιού να είναι εφικτή σε μερικά δευτερόλεπτα. Η επιτυχία της επίθεσης αυτής οφείλεται, αφενός στο υποστήριξη του DHE_EXPORT από τον εξυπηρετητή και την προεργασία για την εύρεση του κλειδιού, αλλά και στην ικανότητα του επιτιθέμενου να κρατήσει την σύνδεση ανοιχτή ως ότου υπολογίσει το κλειδί.

Η επίθεση φαίνεται στην εικόνα 27 και περιγράφεται παρακάτω:

- Ο C στέλνει το ClientHello μήνυμα με ένα από τους αλγόριθμους ανταλλαγής κλειδιών τον DHE.
- Ο A παραποιεί το μήνυμα, αφήνοντας το DHE_EXPORT ως μόνη επιλογή αλγορίθμου ανταλλαγής κλειδιών και το προωθεί στον S.
- Ο S απαντά με το ServerHello και αποδεχόμενος το DHE_EXPORT.
- Ο A τροποποιεί το μήνυμα, τοποθετώντας το DHE στη θέση του DHE_EXPORT και προωθεί το μήνυμα στον C.
- Ο S στέλνει το πιστοποιητικό του υπογεγραμμένο από ένα δυνατό ιδιωτικό κλειδί.
- Ο S στέλνει το ServerKeyExchange μήνυμα με τις παραμέτρους για το 512-bit κλειδί, και το ServerHelloDone.
- Ο A τερματίζει τη σύνδεση προς τον S και ξεκινά τον υπολογισμό του κλειδιού.
- Ο C στέλνει το ClientKeyExchange και στέλνει την επιβεβαίωση του μέσω του ClientCCS και ClientFinished, περιμένοντας την επιβεβαίωση από το εξυπηρετητή.

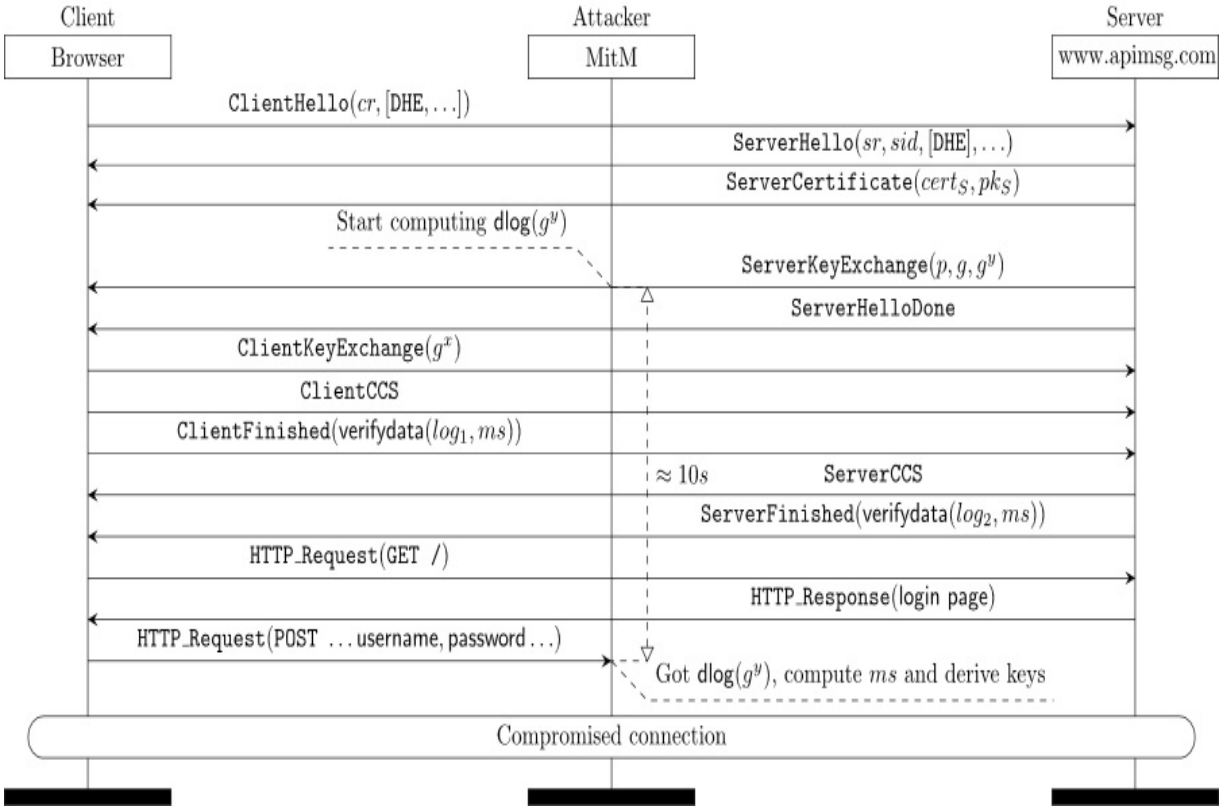
- Ο Α προσπαθεί να κρατήσει την σύνδεση ζωντανή με διάφορα ψεύτικα μηνύματα ειδοποίησης, που αγνοούνται από τον C, και ταυτόχρονα να υπολογίσει το κλειδί.
- Μόλις πραγματοποιηθεί ο υπολογισμός του κλειδιού, ο Α στέλνει τα μηνύματα επιβεβαίωσης ServerCCS και ServerFinished και ακολούθως ο C στέλνει τα δεδομένα στον Α, που είναι συνδεδεμένος.




**Εικόνα 27.** Επίθεση Logjam με υποβιβασμό του αλγορίθμου ανταλλαγής κλειδιών σε DHE_EXPORT και υπολογισμός του κλειδιού κατά τη διάρκεια της σύνδεσης (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

Ο μοναδικός αρνητικός παράγοντας για τον επιτιθέμενο είναι ο χρόνος, καθώς ένας πελάτης μπορεί να τερματίσει αυτός την σύνδεση, αν αυτή καθυστερεί υπέρ του δέοντος. Ακόμα και αν ο ίδιος ο πελάτης δεν τερματίσει τη σύνδεση, αυτό μπορεί να γίνει από το φυλλομετρητή. Η προσπάθεια εύρεσης του κλειδιού έχει χρονομετρηθεί στα 70 δευτερόλεπτα, και ο μόνος φυλλομετρητής που παρέμεινε τόση ώρα σε σύνδεση ήταν ο Firefox.

Τέλος η τελευταία περίπτωση επίθεσης πραγματοποιείται όταν χρησιμοποιούνται κλειδιά μεγέθους 768-bit και 1024-bit με τους πρώτους αριθμούς να είναι «ασφαλείς», και σ' αυτό οφείλεται και η επιτυχία της. Οι πρώτοι αυτοί αριθμοί εξάγονται μέσα από τη σχέση p-1=2q για κάποιο πρώτο αριθμό p. Τότε τα γκρουπ στο Diffie-Hellman μπορούν να πάρουν τις τιμές 2, q ή 2q. Επομένως αν η τιμή g στην εξίσωση του Diffie-Hellman μπορεί να πάρει μόνο αυτές τις περιορισμένες τιμές, η επίλυση του διακριτού λογαρίθμου μπορεί να πάρει την τιμή $\sqrt{q}$ για τους αλγορίθμους baby-step giant-step και Pollard rho.




**Εικόνα 28.** Επίθεση Logjam σε εξυπηρετητή που χρησιμοποιεί «ασφαλείς» πρώτους αριθμούς (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

Όπως φαίνεται στην εικόνα 28 η επίθεση αυτή μοιάζει πολύ με την πρώτη περίπτωση. Ο επιτιθέμενος δεν χρειάζεται να τροποποιήσει κανένα μήνυμα που ανταλλάσσει ο εξυπηρετητής με τον πελάτη. Το μόνο που χρειάζεται είναι ο υπολογισμός του κλειδιού με την παραλαβή των παραμέτρων από τον εξυπηρετητή στο μήνυμα ServerKeyExchange. Η διαδικασία αυτή δεν κρατάει πάνω από δέκα δευτερόλεπτα.

Οι πλείστες από τις παραπάνω επιθέσεις επιτευχθήκαν με τη χρήση ενός εργαλείου που κατασκευάστηκε από μια ομάδα ερευνητών, το FlexTLS, με στόχο τον έλεγχο του πρωτοκόλλου TLS και των αδυναμιών του. Το FlexTLS βασίζεται στη βιβλιοθήκη miTLS. Τα εν λόγω εργαλεία θα αναπτυχθούν στο επόμενο κεφάλαιο (Adrian, D., Bhargavan, K., Durumeric, Z., Caudry, P., Green, M., Halderman, A., J., Heninger, N., Springall, D., Thome, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanela-Beguelin, S., Zimmerman, P., 2015, σσ. 2-8) (Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, A., J., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E, Zanella-Beguelin, S., Zimmermann P., 2015) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

# Κεφάλαιο 5

# Λογισμικά Εργαλεία

Για την πραγματοποίηση επιθέσεων σε ένα ελεγχόμενο περιβάλλον, χρειάστηκε η εγκατάσταση και εκμάθηση ορισμένων λογισμικών εργαλείων. Το πρώτο εργαλείο που εγκαταστάθηκε ήταν το λογισμικό του OpenSSL. Ακολούθως μετά από πολλές προσπάθειες έγινε κατορθωτή και η εγκατάσταση του λογισμικού FlexTLS που χρησιμοποιεί τη βιβλιοθήκη miTLS. Τα εργαλεία είναι ανοιχτού κώδικα και περιγράφονται παρακάτω.

## 5.1 OpenSSL

Όπως έχει αναφερθεί και πιο πάνω, το λογισμικό του OpenSSL είναι ανοιχτού κώδικα, πράγμα που σημαίνει ότι ο οποιοσδήποτε προγραμματιστής μπορεί να δει, να διαβάσει και να επεξεργαστεί τον κώδικά του. Υπεύθυνοι για το OpenSSL είναι οι Eric A. Young και Tim J. Hudson, οι οποίοι για πρώτη φορά ανέπτυξαν το πρόγραμμα το 1995 με την ονομασία SSLeay, και δημοσιεύθηκε αργότερα, το 1998, με την τελική ονομασία OpenSSL.

Το OpenSSL είναι υλοποιημένο σε γλώσσα προγραμματισμού C, και χρησιμοποιείται για να προστατεύει τα δεδομένα που ανταλλάσσουμε στο διαδίκτυο. Είναι επί της ουσίας ένα πλήρες εργαλείο που εφαρμόζεται στα πρωτόκολλα Secure Socket Layer (SSL) και Transport Layer Security (TLS), και περιέχει μια κρυπτογραφική βιβλιοθήκη. Περιέχει πολλούς αλγορίθμους για ανταλλαγή κλειδιών, συμμετρική κρυπτογραφία, για

συναρτήσεις κατακερματισμού, για ψηφιακές υπογραφές και ψηφιακά πιστοποιητικά. Το λογισμικό διαχειρίζονται διάφοροι εθελοντές από όλα τα μέρη του κόσμου. Ο ετήσιος προϋπολογισμός του είναι περί το ένα εκατομμύριο δολάρια Αμερικής, και όλα τα έσοδά του προέρχονται από δωρεές. Μπορεί να χρησιμοποιηθεί για μη-εμπορικούς, αλλά και για εμπορικούς σκοπούς.

Το OpenSSL επικυρώθηκε στο FIPS 140-2 (Evans, D., L., Bond, P., J., Bement, A., L., 2001) για την ασφάλεια υπολογιστών, κάτω από τον οργανισμό National Institute of Standards and Technology's (NIST) και Cryptographic Module Validation Program (CMVP). Είναι διπλά αδειοδοτημένο κατέχοντας την OpenSSL άδεια και την SSLeay άδεια. Η OpenSSL άδεια είναι στην ουσία η άδεια του Apache 1.0, ενώ η SSLeay μοιάζει με τη 4-clause BSD License. Η OpenSSL δεν είναι συμβατή με GPL.

Για το λόγο αυτό έχουν αναπτυχθεί και άλλα λογισμικά ανοιχτού κώδικα, με την ίδια χρήση, για να καλύψουν ανάγκες, που η OpenSSL δεν μπορεί. Η GnuTLS είναι ένα λογισμικό υλοποιημένο σε C με άδεια LPGL. Επίσης υπάρχει η BoringSSL που αναπτύχθηκε και χρησιμοποιείται από την Google. Η JSSE που χρησιμοποιείται από τη Java. Ακόμη υπάρχουν η NSS (Network Security Service), ανοιχτού κώδικα βιβλιοθήκη, η CyaSSL, που βασίζεται στο μέγεθος και την ταχύτητα και πολλές άλλες (OpenSSL, 2015) (Richardson, 2014) (Ristić, I., 2015) (Wikipedia, 2016) (Μωραΐτης, Δ., Σ., 2012, σ. 27).

### 5.1.1 Προηγούμενες Εκδόσεις και Ευπάθειες

Το λογισμικό OpenSSL, βρέθηκε κατά καιρούς κάτω από διάφορες ευπάθειες που οφείλονταν σε προγραμματιστικά κενά (bug), που οδήγησαν σε επιτυχημένες επιθέσεις ενδιαμέσου (Man-In-The-Middle Attack). Για το λόγω αυτό οι εθελοντές που ασχολούνται με το OpenSSL, διόρθωσαν τα σφάλματα αυτά, είτε αναβαθμίζοντας το σε νέες εκδόσεις. Οι κυριότερες ευπάθειες παρουσιάζονται παρακάτω:

- **CVE-2014-0160:** Η επίθεση του Heartbleed, η οποία πραγματοποιήθηκε τον Απρίλιο του 2014, είχε ως αποτέλεσμα την παράκαμψη τις ασφαλούς επικοινωνίας με τον εξυπηρετητή και την υποκλοπή διαπιστευτήρια εισαγωγής (login credentials), προσωπικά δεδομένα και κλειδιά αποκρυπτογράφησης. Οι εκδόσεις που είναι επιρρεπείς στην επίθεση αυτή είναι οι OpenSSL 1.0.1f, 1.0.1e,

1.0.1d, 1.0.1c, 1.0.1b, 1.0.1a, 1.0.1. Το κενό τελικά διορθώθηκε με την ανανεωμένη έκδοση OpenSSL 1.0.1g.

- **CVE-2014-0224:** Τον Ιούλιο του ιδίου χρόνου, είχαμε την επίθεση EarlyCCS, όπου ο επιτιθέμενος έστελνε το CCS μήνυμα, αμέσως μετά το ServerHelloDone οδηγώντας στον υπολογισμό ενός κενού master_secret και τον υπολογισμό αδύναμων κλειδιών κρυπτογράφησης. Αυτό επιτευχθεί στις εκδόσεις του OpenSSL 1.0.1g, 1.0.1f, 1.0.1e, 1.0.1d, 1.0.1c, 1.0.1b, 1.0.1a, 1.0.1, 1.0.0l, 1.0.0k, 1.0.0j, 1.0.0i, 1.0.0g, 1.0.0f, 1.0.0e, 1.0.0d, 1.0.0c, 1.0.0b, 1.0.0a, 1.0.0, 0.9.8y, 0.9.8x, 0.9.8w, 0.9.8v, 0.9.8u, 0.9.8t, 0.9.8s, 0.9.8r, 0.9.8q, 0.9.8p, 0.9.8o, 0.9.8n, 0.9.8m, 0.9.8l, 0.9.8k, 0.9.8j, 0.9.8i, 0.9.8h, 0.9.8g, 0.9.8f, 0.9.8e, 0.9.8d, 0.9.8c, 0.9.8b, 0.9.8a, 0.9.8. Διορθώθηκε δε στις εκδόσεις OpenSSL 1.0.1h, 1.0.0m, 0.9.8za.
- **CVE-2014-3566:** Μετά από λίγο καιρό, τον Οκτώβριο του 2014, η επίθεση POODLE που υλοποιείτο στο πρωτόκολλο SSL v3.0. Η υποβίβαση του πρωτοκόλλου ασφαλείας στην πεπαλαιωμένη και απαρχαιωμένη έκδοση του SSL v3.0 ήταν επιτυχείς, λόγω του γεγονότος ότι οι φυλλομετρητές και οι εξυπηρετητές υποστήριζαν ακόμη την έκδοση αυτή και λόγω του ότι το OpenSSL υποστήριζε το TLS_FALLBACK_SCSV. Οι επιρρεπείς εκδόσεις είναι οι OpenSSL 1.0.1i, 1.0.1h, 1.0.1g, 1.0.1f, 1.0.1e, 1.0.1d, 1.0.1c, 1.0.1b, 1.0.1a, 1.0.1, 1.0.0n, 1.0.0m, 1.0.0l, 1.0.0k, 1.0.0j, 1.0.0i, 1.0.0g, 1.0.0f, 1.0.0e, 1.0.0d, 1.0.0c, 1.0.0b, 1.0.0a, 1.0.0, 0.9.8zb, 0.9.8za, 0.9.8y, 0.9.8x, 0.9.8w, 0.9.8v, 0.9.8u, 0.9.8t, 0.9.8s, 0.9.8r, 0.9.8q, 0.9.8p, 0.9.8o, 0.9.8n, 0.9.8m, 0.9.8l, 0.9.8k, 0.9.8j, 0.9.8i, 0.9.8h, 0.9.8g, 0.9.8f, 0.9.8e, 0.9.8d, 0.9.8c, 0.9.8b, 0.9.8a, 0.9.8. Η ευπάθεια διορθώθηκε στις εκδόσεις OpenSSL 1.0.1j, 1.0.0o και 0.9.8zc.
- **CVE-2015-0204:** Το Μάρτιο του 2015 είχαμε την επίθεση του FREAK, όπου ο επιτιθέμενος μπορούσε να υποβιβάσει μια σύνδεση που υποστήριζε RSA για αλγόριθμο ανταλλαγής κλειδιών στον αποσυρόμενο RSA_EXPORT με μήκος κλειδιού τα 512-bit, δίνοντας του την δυνατότητα να διασπάσει το κλειδί. Οι ευπαθείς εκδόσεις είναι OpenSSL 1.0.1j, 1.0.1i, 1.0.1h, 1.0.1g, 1.0.1f, 1.0.1e, 1.0.1d, 1.0.1c, 1.0.1b, 1.0.1a, 1.0.1, 1.0.0o, 1.0.0n, 1.0.0m, 1.0.0l, 1.0.0k, 1.0.0j, 1.0.0i, 1.0.0g, 1.0.0f, 1.0.0e, 1.0.0d, 1.0.0c, 1.0.0b, 1.0.0a, 1.0.0, 0.9.8zc, 0.9.8zb, 0.9.8za, 0.9.8y, 0.9.8x, 0.9.8w, 0.9.8v, 0.9.8u, 0.9.8t, 0.9.8s, 0.9.8r, 0.9.8q, 0.9.8p, 0.9.8o, 0.9.8n, 0.9.8m, 0.9.8l, 0.9.8k, 0.9.8j, 0.9.8i, 0.9.8h, 0.9.8g, 0.9.8f, 0.9.8e, 0.9.8d, 0.9.8c, 0.9.8b, 0.9.8a, 0.9.8. Η επίθεση αποτράπηκε στις εκδόσεις OpenSSL 1.0.1k, 1.0.0p, 0.9.8zd.

- **CVE-2015-0291:** Τον ίδιο μήνα ανακαλύφθηκε και μια ευπάθεια που οδηγούσε σε επίθεση άρνησης εκτέλεσης εντολών (Denial-of-Service (DoS)). Η επίθεση αφορούσε την έκδοση OpenSSL 1.0.2, και διορθώθηκε στην έκδοση OpenSSL 1.0.2a.
- **CVE-2015-4000:** Το Μάιο του 2015, είχαμε την επίθεση Logjam, που θυμίζει την FREAK, και οδηγεί στην υποβάθμιση του αλγορίθμου DHE σε DHE_EXPORT με κλειδί μήκους 512-bit. Οι εκδόσεις που παρουσίαζαν ευπάθεια είναι OpenSSL 1.0.2a ,1.0.2, 1.0.1m, 1.0.1l, 1.0.1k, 1.0.1j, 1.0.1i, 1.0.1h, 1.0.1g, 1.0.1f, 1.0.1e, 1.0.1d, 1.0.1c, 1.0.1b, 1.0.1a, 1.0.1. Οι διορθωμένες εκδόσεις είναι οι OpenSSL 1.0.2b, 1.0.1n.
- **CVE-2015-1793:** Τον Ιούλη του ιδίου έτους, η πλαστογραφία εναλλακτικών αλυσίδων πιστοποίησης (Alternative Chains Certificate Forgery), με την οποία ο επιτιθέμενος εξανάγκαζε το φυλλομετρητή να αποδεχτεί πιστοποιητικό από μια μη πιστοποιημένη Αρχή Πιστοποίησης (CA). Αυτό επηρεάζει τις εκδόσεις OpenSSL 1.0.2c, 1.0.2b, 1.0.1n, and 1.0.1o, και διορθώνεται στις εκδόσεις OpenSSL 1.0.2d, 1.0.1p (OpenSSL, 2015) (Richardson, 2014) (Symantec, 2015).

### 5.1.2 Δημιουργία Ψηφιακών Πιστοποιητικών X.509

Το λογισμικό του OpenSSL, όπως είπαμε είναι ένα πολύ χρήσιμο εργαλείο για το TLS και SSL, με το οποίο μπορείς να κατασκευάσεις και πιστοποιητικά. Η κατασκευή των πιστοποιητικών γίνεται μέσω της γραμμής εντολών και μπορεί να γίνει απ' ευθείας με μια μόνο εντολή, αλλά μπορεί να γίνει και σε τρία βήματα. Πριν από αυτό θα ελέγξουμε την έκδοση του OpenSSL για να δούμε αν η έκδοσή του είναι επιρρεπείς σε οποιανδήποτε ευπάθειες και χρειάζεται αναβάθμιση. Παρακάτω θα αναλυθούν ορισμένες βασικές εντολές με τα παραδείγματά τους για καλύτερη κατανόηση που βρίσκονται στο βιβλίο (Ristić, I., 2015).

Για τη λήψη και εγκατάσταση του OpenSSL χρησιμοποιούμε με τη σειρά τις παρακάτω εντολές.

```
$ wget http://www.openssl.org/source/openssl-1.0.1p.tar.gz
$ ./config \ --prefix=/opt/openssl \ --openssldir=/opt/openssl \ enable-ec_nistp_64_gcc_128
$ make depend
```

*$ make*

*$ sudo make install*

Αφού εγκατασταθεί μπορούμε να το ελέγξουμε ποια έκδοση εγκαταστήσαμε. Η έκδοση ελέγχεται με την εντολή:

*$ openssl version*

το αποτέλεσμα της είναι:

OpenSSL 1.0.1 14 Mar 2012

και μπορούμε να ξεχωρίσουμε την έκδοση, η οποία είναι η 1.0.1 και δίπλα αναγράφεται η ημερομηνία δημοσιοποίησής της. Με την εντολή:

*$ openssl version –a*

μπορούμε να πάρουμε περισσότερες πληροφορίες που αφορούν την έκδοση.

OpenSSL 1.0.1 14 Mar 2012
built on: Fri Jun 20 18:54:15 UTC 2014
platform: debian-amd64
options:  bn(64,64) rc4(8x,int) des(idx,cisc,16,int) blowfish(idx)
compiler: cc -fPIC -DOPENSSL_PIC -DZLIB -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN ↩
-DHAVE_DLFCN_H -m64 -DL_ENDIAN -DTERMIO -g -O2 -fstack-protector --param=ssp-buffer-size↩
=4 -Wformat -Wformat-security -Werror=format-security -D_FORTIFY_SOURCE=2 ↩
-Wl,-Bsymbolic-functions -Wl,-z,relro -Wa,--noexecstack -Wall -DOPENSSL_NO_TLS1_2↩
_CLIENT -DOPENSSL_MAX_TLS1_2_CIPHER_LENGTH=50 -DMD32_REG_T=int -DOPENSSL_IA32_SSE2 ↩
-DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256↩
_ASM -DSHA512_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DWHIRLPOOL_ASM -DGHASH↩
_ASM
OPENSSLDIR: "/usr/lib/ssl"

Η τελευταία γραμμή μας δηλώνει το μονοπάτι που είναι φυλαγμένο στο δίσκο το OpenSSL.

Μπορείς να πάρεις βοήθεια για τις εντολές που μπορείς να χρησιμοποιήσεις με μια λάθος εντολή:

*$ openssl help*

το αποτέλεσμα θα είναι:

openssl:Error: 'help' is an invalid command.

```
Standard commands
asn1parse     ca            ciphers       cms
crl           crl2pkcs7     dgst          dh
dhparam       dsa           dsaparam      ec
ecparam       enc           engine        errstr
gendh         gendsa        genpkey       genrsa
nseq          ocsp          passwd        pkcs12
pkcs7         pkcs8         pkey          pkeyparam
pkeyutl       prime         rand          req
rsa           rsautl        s_client      s_server
s_time        sess_id       smime         speed
spkac         srp           ts            verify
version       x509
```

Για περισσότερες πληροφορίες για κάποια εντολή μπορείς να χρησιμοποιήσεις την εντολή man. Για παράδειγμα man x509, θα σου δώσει περισσότερες πληροφορίες για τις παραμέτρους που μπορείς να συμπληρώσεις για να δημιουργήσεις το κατάλληλο πιστοποιητικό.

Για τη δημιουργία πιστοποιητικού θα πρέπει να δημιουργήσεις πρώτα ένα κλειδί. Το κλειδί μπορεί να δημιουργηθεί με τους αλγόριθμους RSA, DSA, ECDSA. Επιπλέον πρέπει να καθοριστεί το μέγεθος του κλειδιού 512, 768, 1024, 2048 ή 4096-bit. Τέλος μπορείς να προστατέψεις το δημιουργημένο κλειδί με κάποιο αλγόριθμο συμμετρικής κρυπτογράφησης π.χ. aes128. Η εντολή έχει ως εξής:

*$ openssl genrsa -aes128 -out fd.key 2048*

θα μας δώσει το:

```
Generating RSA private key, 2048 bit long modulus
....+++
.....................................................................+++
e is 65537 (0x10001)
Enter pass phrase for fd.key: ****************
Verifying - Enter pass phrase for fd.key: ****************
```

Η εισαγωγή κωδικού πρέπει να γίνει με πολύ προσοχή, γιατί αν ξεχαστεί τότε δε θα μπορεί να ανακληθεί το κλειδί και να δημιουργηθεί το πιστοποιητικό. Η προστασία με συμμετρική κρυπτογράφηση μπορεί να αποφευχθεί σε περίπτωση που είμαστε σίγουροι ότι το κλειδί εκεί που φυλάσσεται είναι ασφαλές.

Με την επόμενη εντολή δημιουργείς ένα προς υπογραφή πιστοποιητικό (Certificate Signing Request (CSR)), βάση του κλειδιού που δημιουργήσαμε προηγουμένος:

```
$ openssl req -new -key fd.key -out fd.csr
  Enter pass phrase for fd.key: ****************
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:GB
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:London
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Feisty Duck Ltd
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:www.feistyduck.com
Email Address []:webmaster@feistyduck.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Σε πρώτη φάση θα ζητηθεί το κλειδί συμμετρικής κρυπτογραφίας που ορίσαμε στη δημιουργία του RSA κλειδιού. Αν δεν ορίστηκε κανένα κλειδί, αυτό το βήμα θα παραλειφθεί. Στη συνέχεια θα ζητηθεί να συμπληρωθούν ορισμένοι παραμέτροι όπως η χώρα, η πόλη, το όνομα της εταιρίας. Μεγάλη προσοχή πρέπει να δοθεί στο πεδίο του ονόματος (Common Name). Αυτό είναι και το όνομα του ισοτόπου. Ένας φυλλομετρητής, θα ελέγξει το όνομα του ισοτόπου αν είναι αυτό που αναγράφεται και στο πιστοποιητικό και αν δε συμβαδίζουν, τότε θα στείλει προειδοποιητικό μήνυμα στο χρήστη. Τέλος συμπληρώνεται το ηλεκτρονικό ταχυδρομείο (email) για σκοπούς επικοινωνίας του χρήστη αν αντιμετωπίζει κάποιο πρόβλημα. Προαιρετικά μπορεί να συμπληρωθούν τα επιπλέον χαρακτηριστικά. Τώρα το πιστοποιητικό πρέπει να

υπογραφεί από κάποια πιστοποιημένη Αρχή Πιστοποίησης (CA) ή να αυτό-υπογραφεί (self-signed) στο OpenSSL, με την εντολή:

*$ openssl x509 -req -days 365 -in fd.csr -signkey fd.key -out fd.crt*

η εντολή παίρνει ως είσοδο το προς υπογραφή πιστοποιητικό, μαζί με το κλειδί. Επιπλέον καθορίζονται οι μέρες που θα είναι έγκυρο το πιστοποιητικό. Ένα αυτό-υπογεγραμμένο πιστοποιητικό, θα δημιουργήσει μήνυμα προειδοποίησης όταν ληφθεί από το φυλλομετρητή, γιατί προέρχεται από μια μη πιστοποιημένη Αρχή Πιστοποίησης (Ristić, I., 2015).

## 5.2 FlexTLS

Το FlexTLS είναι εργαλείο που χρησιμοποιείται για τον έλεγχο του πρωτοκόλλου TLS. Ενώ αρχικά είχε αναπτυχθεί στην προσπάθεια να επιτευχθούν περίπλοκες επιθέσεις στο TLS, όπως αυτή του Triple Handshake και του EarlyCCS στο OpenSSL, το εργαλείο αναπτύχθηκε και επεκτάθηκε με σκοπό την εκτέλεση διάφορων σεναρίων. Έχει την δυνατότητα σύνδεσης, είτε ως πελάτης, είτε ως εξυπηρετητής για να συντάσσει και να ανταλλάσσει μηνύματα του πρωτοκόλλου TLS. Αυτό βέβαια γινόταν ελεγχόμενα με σκοπό τον έλεγχο της ορθότητας του πρωτοκόλλου και την ανακάλυψη τυχόν κενών ασφαλείας και ευπαθειών. Η παραπάνω πρακτική οδήγησε στην ανακάλυψη πολύ επικίνδυνων ευπαθειών, όπως το SKIP Attack, το FREAK και το Logjam.

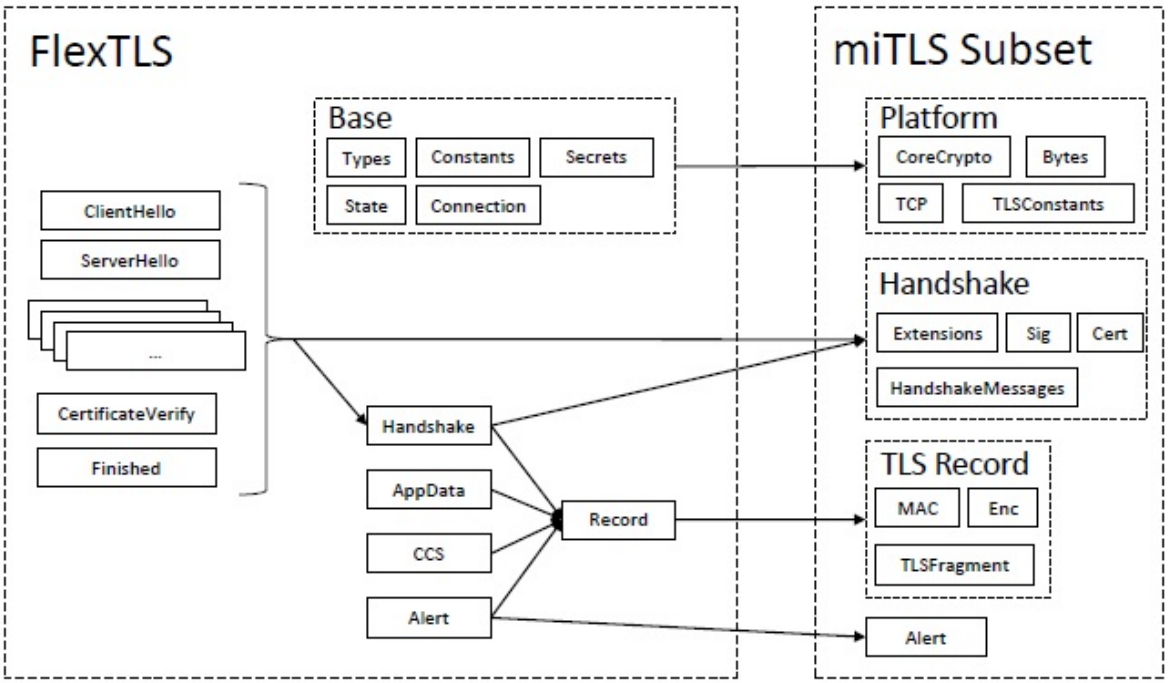
Οι παραπάνω ευπάθειες οδήγησαν το IETF, στη σκέψη για επανασχεδιασμό του πρωτοκόλλου TLS στην έκδοση 1.3 και την προσθήκη επεκτάσεων και καινούργιων μηνυμάτων. Το FlexTLS είναι ένα εργαλείο σχεδιασμένο γι' αυτό το σκοπό, για τον έλεγχο και ανάλυση του πρωτοκόλλου. Έχει σχεδιαστεί με σκοπό να μπορεί να υποστηρίξει νέα μοντέλα, όπως είναι το TLS v1.3, γιατί μπορεί να εφαρμοστεί γρήγορα και να αξιολογήσει νέες προτάσεις.

Το FlexTLS είναι δομημένο βάση της βιβλιοθήκης miTLS, που είναι μια επικυρωμένη εφαρμογή του TLS. Επαναχρησιμοποιεί τα στοιχεία για τη δομή των μηνυμάτων του miTLS και των κρυπτογραφικών αλγορίθμων, με τέτοιο τρόπο ώστε να του προσφέρει ευελιξία και επιτρέποντας του να χρησιμοποιηθούν με νέους απρόβλεπτους τρόπους. Με αυτό τον τρόπο, αφού τα στοιχεία των μηνυμάτων του miTLS έχουν αποδείξει την

ορθότητα τους, τότε μπορούμε να πούμε ότι και τα στοιχεία των μηνυμάτων στο FlexTLS είναι ορθά.

Το miTLS είναι μια πολύ αυστηρή διεπαφή προγραμματισμού εφαρμογών (application programming interface), το οποίο εγγυάται την αποστολή και παράδοση των μηνυμάτων βάση των προδιαγραφών του πρωτοκόλλου, χωρίς παρεκκλίσεις, εν αντιθέσει με το FlexTLS, το οποίο έχει σχεδιαστεί με σκοπό την ευελιξία στην εισαγωγή νέων μηνυμάτων, αναδιάταξη της σειρά τους και στο πειραματισμό με νέα πρωτόκολλα. Το FlexTLS είναι γραμμένο σε γλώσσα προγραμματισμού F# και δομημένο σε .NET βιβλιοθήκη, επομένως οι χρήστες μπορούν να γράψουν και να εξετάσουν διάφορα σενάρια γραμμένα σε οποιανδήποτε .NET γλώσσα προγραμματισμού. Η διεπαφή προγραμματισμού εφαρμογών έχει τα παρακάτω χαρακτηριστικά:

- Παροχή υψηλού επιπέδου ανταλλαγή μηνυμάτων με λογικές προεπιλογές.
- Διαχείριση ταυτόχρονα πολλαπλών συνδέσεων.
- Υποστηρίζει την αυθόρμητη αναδιάταξη, κατακερματισμό και αλλοίωση των μηνυμάτων.
- Οι επεκτάσεις του miTLS, ευνοούν χαρακτηρίστηκα νέων πρωτοκόλλων.




**Εικόνα 29.** Η αρχιτεκτονική του FlexTLS (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015).

Στην εικόνα 29 απεικονίζεται η αρχιτεκτονική του FlexTLS, όπου κάθε πρωτόκολλο (Handshake, AppData κτλ.) και κάθε μήνυμα (ClientHello κτλ.), που εφαρμόζεται στο λογισμικό του FlexTLS, καλείται κατ' ευθείαν από τη βιβλιοθήκη του miTLS, που βρίσκεται αριστερά, για να υλοποιηθεί. Επίσης τα μηνύματα και τα πρωτόκολλα είναι δομημένα ως στοιχεία, με μοναδικό στόχο την αποστολή και παραλαβή μηνυμάτων, έτσι ώστε το FlexTLS να μπορεί να τα ελέγξει σε διάφορα επίπεδα. Για παράδειγμα, το ClientHello μπορεί να ελεγχθεί από το χρήστη σε υψηλό επίπεδο για την παραγωγή σωστών Hello μηνυμάτων, ή μπορεί να σε χαμηλό επίπεδο, αυτό του Handshake, με τροποποίηση των bytes. Ενώ οι συναρτήσεις που υλοποιούνται στο FlexTLS, θα προσπαθήσουν να χρησιμοποιήσουν τις λογικές προεπιλογές, λόγω της υλοποίηση του στη F# θα μας δώσει διαφορετικό αποτέλεσμα.

Κάθε συνάρτηση στο FlexTLS είναι γραμμένη σε μια πάρε-δώσε λειτουργική κατάσταση, πράγμα που σημαίνει ότι κάθε συνάρτηση παίρνει μια είσοδο και δίνει μια έξοδο, χωρίς να συντηρεί εσωτερικές καταστάσεις, με μόνο μέλημα την αποστολή και παραλαβή TCP μηνυμάτων. Αυτό κάνει το FlexTLS να ξεχωρίζει σε σχέση με άλλα εργαλεία που μπορούν να υλοποιήσουν TLS συνδέσεις, όπως το OpenSSL, όπου είναι δύσκολη η αναδιάταξη μηνυμάτων και επαναφορά της σύνδεσης σε ένα πρωτύτερο σημείο. Οι διαφορετικές συνδέσεις δεν επηρεάζουν η μια την άλλη, με αποτέλεσμα, έτσι τα διάφορα σενάρια μπορούν να ξεκινήσουν συνδέσεις ως πελάτες ή ως εξυπηρετητές, παίρνοντας παραμέτρους, όπως το session_id, από τη μια σύνδεση στην άλλη, επαναφέροντας τη σύνδεση σε πρωτύτερο σημείο ή αποδεσμεύοντας την εντελώς αν δεν χρειάζεται πλέον.

Το FlexTLS δεν σταματά απρόσκοπτα τη λειτουργία του για τρείς βασικούς λόγους:

I. Είναι γραμμένο σε γλώσσα προγραμματισμού F#, και έτσι αποφεύγονται σφάλματα υπερχείλισης (buffer overruns).
II. Η χρήση των συναρτήσεων, που δε συντηρούν εσωτερική κατάσταση, αποφεύγονται σφάλματα εκτέλεσης (runtime error), στη ταυτόχρονη τροποποίηση καταστάσεων.
III. Κληρονομά από το miTLS την ορθότητα και την ασφάλεια για την κωδικοποίηση και αποκωδικοποίηση μηνυμάτων και τη χρήση των κρυπτογραφικών συναρτήσεων.

Το FlexTLS προσφέρει ένα εύκολο τρόπο για την επέκταση του miTLS, αφού μπορεί να εξετάσει και να επικυρώσει καινούργια χαρακτηρίστηκα του πριν ενσωματωθούν στο miTLS (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015, pp. 3-4) (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015, pp. 1-3) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

## 5.3 miTLS

Η βιβλιοθήκη miTLS αποτελεί μια εφαρμογή του πρωτοκόλλου TLS, υποστηρίζοντας όλους τους κρυπταλγόριθμους, τις συνόδους, συνδέσεις, χειραψίες, τα σφάλματα, τις προειδοποιήσεις και το κατακερματισμό δεδομένων, όπως περιγράφονται στα RFCs, και είναι συμβατό με όλους τους γνωστούς φυλλομετρητές και εξυπηρετητές. Λόγω του ότι ο κώδικας είναι δομημένος κατά στοιχεία, επικυρώνεται αυτόματα, από το κυρίως πρόγραμμα μέχρι το επίπεδο υπολογισμού των κρυπτογραφικών αλγορίθμων.

Η έκδοση του miTLS είναι η 0.9, και είναι γραμμένο σε γλώσσα προγραμματισμού F#, και ορίζεται λεπτομερώς στη F7, για τα κύρια συστατικά της χειραψίας (handshake), που είναι η ανταλλαγή κλειδιού και για το για το πρωτόκολλο Record Layer, που είναι η αυθεντικοποίηση του μηνύματος. Κάθε αλγόριθμος κρυπτογράφησης, είναι εξοπλισμένος με νέες ιδιότητες ασφαλείας, ενώ η αντικαταστάθηκε η συμπαγής τους μορφή με εξιδανικευμένες λειτουργίες. Τέλος, το miTLS, μπορεί να χρησιμοποιηθεί για τον έλεγχο κάποιων θεωριών ασφαλείας γύρω από το πρωτόκολλο ασφαλείας που αναπτύχθηκαν και εφαρμόστηκαν. Αυτό μπορεί να γίνει κατορθωτό με τον έλεγχο, παλιών αλλά και καινούργιων επιθέσεων (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015, p. 1) (Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K., 2015, p. 1) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

# Κεφάλαιο 6

# Μοντελοποίηση των Επιθέσεων

Σκοπός της παρούσας Διπλωματικής είναι η κατανόηση της ασφάλειας στο διαδίκτυο, και η ανάδειξη διάφορων ευπαθειών των πρωτοκόλλων ασφαλείας, μέσω διάφορων επιθέσεων σε αυτά. Για το λόγο αυτό αναπτύχθηκε ένα πειραματικό περιβάλλον εργασίας, με σκοπό τον πειραματισμό και την εκδήλωση ελεγχόμενων επιθέσεων. Σ' αυτό το κεφάλαιο θα γίνει μια εκτενής αναφορά και περιγραφή του περιβάλλοντος εργασίας , με τα εργαλεία που χρησιμοποιήθηκαν. Ακολούθως θα αναλυθεί η υλοποίηση των επιθέσεων που κατάφεραν να επιτευχθούν, και τέλος θα σχολιαστούν τα αποτελέσματα αυτών κάνοντας μια γενική αποτίμηση και σύγκριση αυτών.

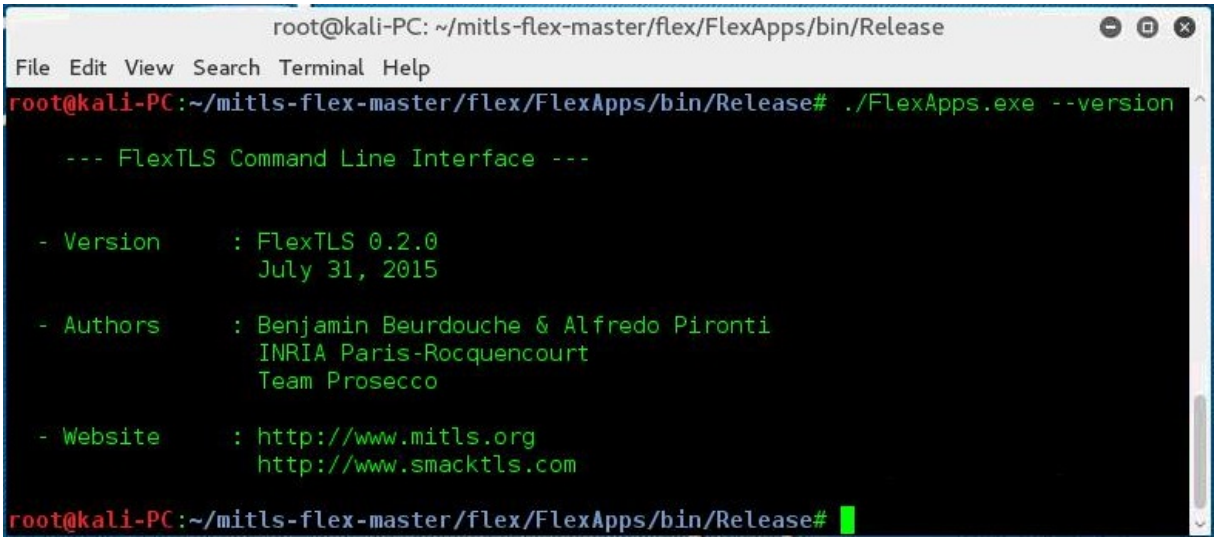
## 6.1 Το Περιβάλλον Εργασίας

Πρώτο βήμα για τη δημιουργία του περιβάλλοντος εργασίας ήταν η εγκατάσταση του εργαλείου FlexTLS σε κάποιο υπολογιστή. Μετά από αρκετές αποτυχημένες προσπάθειες έγινε κατορθωτή η εγκατάστασή του σε ένα υπολογιστή που τρέχει λειτουργικό Kali Linux 2.0. Το λειτουργικό αυτό είναι το πιο γνωστό για την υλοποίηση επιθέσεων στο διαδίκτυο ή σε τοπικά δίκτυα.

Η εγκατάσταση του FlexTLS είχε κάποιες προ απαιτούμενες εγκατεστημένες εφαρμογές για την ορθή εγκατάστασή του. Το λειτουργικό σύστημα του Kali Linux έχει ένα πλήθος εφαρμογών, ήδη εγκατεστημένες ως προεπιλογή μαζί με την εγκατάστασή του. Δυο από αυτές ήταν αναγκαίες για την εγκατάσταση του FlexTLS. Η εφαρμογή του OpenSSL έκδοση 1.0.2f και επιπλέον η εφαρμογή έκδοση python 2.7.11. Ακολούθως έπρεπε να κατεβάσω και να εγκαταστήσω τη γλώσσα προγραμματισμού F# έκδοση 3.1 από τον ισότοπο http://fsharp.org/. Στη συνέχεια κατέβασα και εγκατέστησα τη βιβλιοθήκη

Mono έκδοση 3.4.0 από το http://www.mono-project.com/download/#download-lin. Αφού εγκατασταθεί το Mono μπορώ εύκολα με μια εντολή να εγκαταστήσω το Mono-Develop. Το Mono-Develop, είναι προαιρετικό, καθώς δεν είναι επί της ουσίας άμεσα προαπαιτούμενο για το FlexTLS. Είναι παρόμοιο με το Visual Studio της Microsoft, και το χρησιμοποίησα για να κατεβάσω κάποια nugget που έλειπαν. Τέλος κατέβασα το κώδικα του προγράμματος FlexTLS από το https://github.com/mitls/mitls-flex, και το έκανα build στο Mono-Develop. Μετά το build με την εντολή make, έγινε εγκατάσταση του FlexTLS και δημιουργήθηκε ένα εκτελέσιμο του προγράμματος στο μονοπάτι mitls-flex-master/flex/FlexApps/bin/Release. Η έκδοση του FlexTLS είναι η 0.2.0 και απεικονίζεται στην εικόνα 30, και δίνεται από την εντολή:

*./FlexApps.exe - -version*




**Εικόνα 30.** Η έκδοση του FlexTLS.

Το εργαλείο FlexTLS έχει πολλές δυνατότητες. Οι δυνατότητες αυτές φαίνονται στην εικόνα 31, με την αναγραφή της εντολής

*./FlexApps.exe - -help*

ή κάνοντας οποιοδήποτε άλλο λάθος στην πιο πάνω γραμμή. Η βασική σύνταξη της εντολής είναι η παρακάτω:

*./FlexApps.exe -s { "το σενάριο που θα εκτελεστεί" } - -connect [ "που θα ενωθεί" ]*

όπου το σενάριο που θα εκτελεστεί επιλέγεται από ένα πλήθος σεναρίων που προσφέρονται από την απλή χειραψία (full handshake) μέχρι την επίθεση logjam. Στο δεύτερο κομμάτι της εντολής καθορίζεται που θα συνδεθεί το FlexTLS, σε πελάτη ή σε εξυπηρετητή. Επιπλέον με τις εντολές - -client-cert και - -server-cert μπορεί να

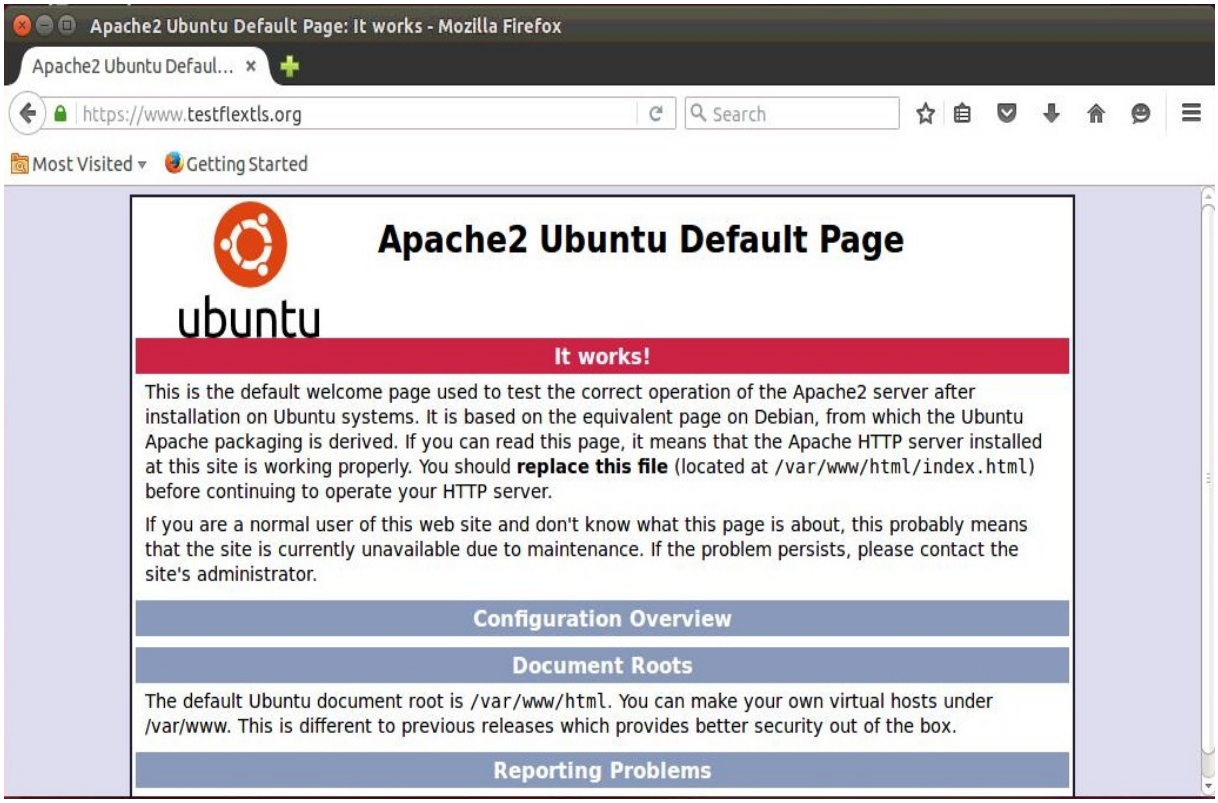
αυθεντικοποιηθεί σαν πελάτης ή σαν εξυπηρετητής, αντίστοιχα. Τέλος για τη πλήρη χειραψία, μπορείς να επιλέξεις μεταξύ τριών ρόλων, (πελάτη (c), εξυπηρετητή (s) και ενδιάμεσου(m)). Επίσης μπορείς να επιλέξεις και τον αλγόριθμο ανταλλαγής κλειδιού μεταξύ του (RSA, DHE, ECDHE).

```
root@kali-PC: ~/mitls-flex-master/flex/FlexApps/bin/Release
File  Edit  View  Search  Terminal  Help
root@kali-PC:~/mitls-flex-master/flex/FlexApps/bin/Release# ./FlexApps.exe --help

    --- FlexTLS Command Line Interface ---

  -h --help     :    This help message
  -v --version  :    Infos about this software

  -s --scenario : *  Scenario to execute
                     - Full Handshake                {fh}
                     - Script.fs                     {script}
                     - SmackTLS                      {smacktls}
                     - Attacks
                         Warning alert               {wal,warningalert}
                         Fragmented alert            {fal,fragmentedalert}
                         Fragmented Client Hello     {fch,fragmentedch}
                         Early CCS                   {eccs,earlyccs}
                         Early CCS (MITM)            {eccsm,earlyccsmitm}
                         Early Finished SKIP         {efin,earlyfinished}
                         Triple Handshake (MITM)     {ths,triplehandshake}
                         Small Subgroup              {sgp,smallsubgroup}
                         Early Resume                {eres,earlyresume}
                         Logjam                      {lj,logjam}
                     - Tests
                         Early AppData               {eapp,earlyappdata}
                         Early AppData after Renego. {eappr,earlyappdatarenego}
  --connect     : [] Connect to address (or domain) and port    (default : localhost:443)
  --client-cert : [] Use client authentication with the given CN
  --accept      : [] Accept address (or domain) and port        (default : localhost:4433)
  --server-cert : [] Certificate CN to use if Server            (default : rsa.cert-01.mitls.org)
  --delay       : [] Wait a delay between SmackTLS scenarios    (default : 0ms)

  *** Options for full handshake scenario ***
  -r --role     : []  Role
                     - Client                        {c,C,Client}  (default)
                     - Server                        {s,S,Server}
                     - Both                          {m,M,MITM}
  -k --kex      : []  Key exchange
                     - RSA                           {rsa,RSA}   (default)
                     - DHE                           {dhe,DHE}
                     - ECDHE                         {ec,ecdhe,ECDHE}

 *  =>  Require an argument
 []  =>  Default will be used if not provided

root@kali-PC:~/mitls-flex-master/flex/FlexApps/bin/Release#
```

**Εικόνα 31.** Όλες οι εντολές που μπορούν να χρησιμοποιηθούν στο FlexTLS.

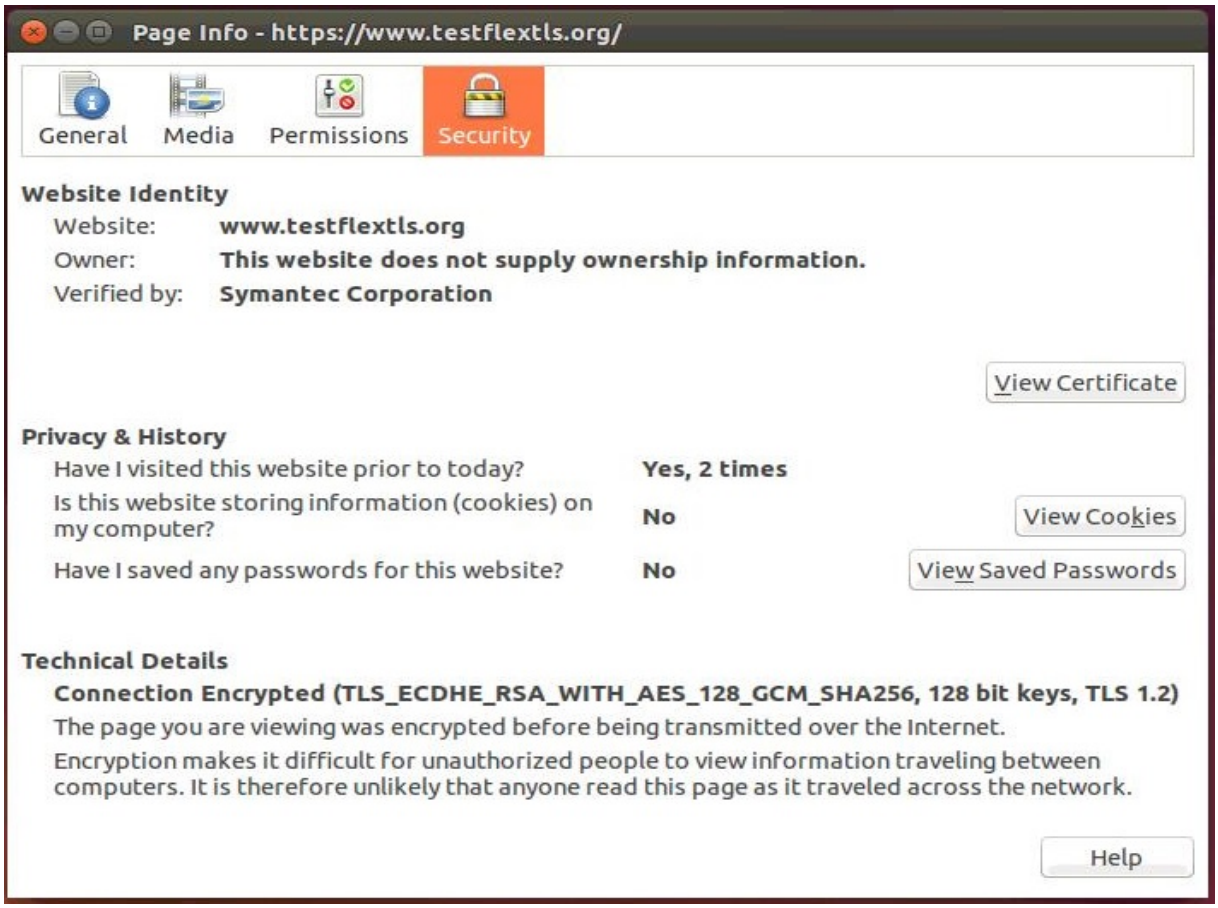
Μετά την επιτυχή εγκατάσταση του FlexTLS, προχώρησα στο δεύτερο μέρος της υλοποίησης των επιθέσεων, που ήταν η δημιουργία ενός εξυπηρετητή ιστού (Web Server). Για να επιτευχθεί αυτό εγκατέστησα σε ένα άλλο υπολογιστή το λειτουργικό σύστημα Ubuntu 14.04 lts και ακολούθως εγκατέστησα τον Apache Server, για εξυπηρετητή ιστού, στη μηχανή. Για τη δημιουργία πιστοποιητικού χρησιμοποίησα την

OpenSSL που ήταν ήδη προ εγκατεστημένη με το λειτουργικό. Η έκδοση του OpenSSL είναι παλιά (1.0.1f) και μη-ενημερωμένη και αυτό βοηθά στο σχεδιασμό του όλου συστήματος για τον έλεγχο ευπαθειών και την υλοποίηση επιθέσεων. Η ιστοσελίδα που διαμοιράζεται ο εξυπηρετητής είναι η προεπιλεγμένη ιστοσελίδα του Apache με κάποιες μικρές αλλαγές, και απεικονίζεται στην εικόνα 30.




**Εικόνα 32.** Η ιστοσελίδα του εξυπηρετητή Apache.

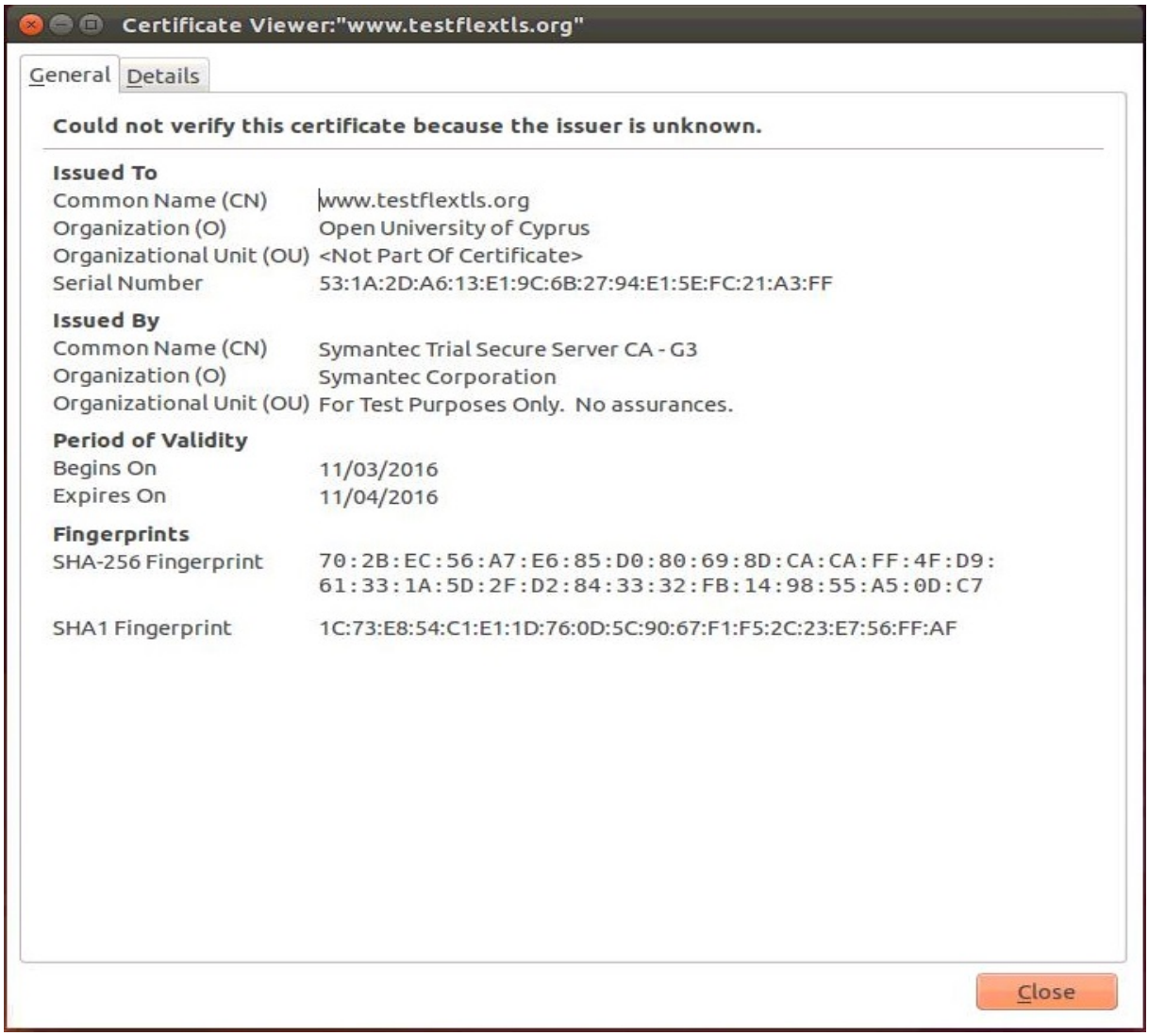
Όπως βλέπουμε και στην εικόνα 32, πάνω στα αριστερά, δίπλα από τη γραμμή όπου εισάγουμε τη διεύθυνση, υπάρχει μια πράσινη κλειδωνιά. Αυτό φανερώνει, ότι η σύνδεση μας με τον εξυπηρετητή είναι ασφαλής και κρυπτογραφημένη. Κάνοντας κλικ στην κλειδωνιά, όπως φαίνεται και στην εικόνα 33, μπορούμε να πάρουμε περισσότερες πληροφορίες. Πρώτα μπορούμε ποιο πρωτόκολλο ασφαλείας και ποια έκδοση του χρησιμοποιεί. Στην προκειμένη περίπτωση χρησιμοποιώ TLS v1.2. Επιπλέον παίρνουμε πληροφορίες για τους αλγόριθμους κρυπτογράφησης που χρησιμοποιούνται. Ο ECDHE χρησιμοποιείται για την ανταλλαγή του κλειδιού, ενώ ο RSA χρησιμοποιείται στην ψηφιακή υπογραφή. Ο αλγόριθμος συμμετρικής κρυπτογραφίας είναι ο AES_128 σε συνδυασμό με το ρυθμό λειτουργίας του GCM. Τέλος η συνάρτηση κατακερματισμού είναι η SHA256 και τα κομμάτια των δεδομένων από τον κατακερματισμό, έχουν μέγεθος 128-bits.




**Εικόνα 33.** Πληροφορίες για την ασφάλεια της σύνδεσης.

Κάνοντας κλικ στο κουμπί που γράφει "View Certificate", μπορούμε να δούμε και να εξετάσουμε το πιστοποιητικό. Για τις ανάγκες της Μεταπτυχιακής Διατριβής, αρχικά είχα δημιουργήσει ένα αυτό-υπογεγραμμένο πιστοποιητικό στο OpenSSL. Παρόλα αυτά, υπήρχε πρόβλημα συμβατότητας με το FlexTLS. Έτσι έστειλα ένα πιστοποιητικό προς υπογραφή στη Symantec (Symantec, 2016) για τη δημιουργία ενός υπογεγραμμένου πιστοποιητικό για δοκιμαστική περίοδο των 30 ημερών.

Το πιστοποιητικό φαίνεται στην εικόνα 34. Εκεί, βλέπουμε διάφορες πληροφορίες που μας δίδει το πιστοποιητικό, όπως το όνομα του ισοτόπου και τον του ισοτόπου. Επιπλέον, παρακάτω, παρατηρούμε ότι το πιστοποιητικό εκδίδεται από τη Symantec. Στο επόμενο πεδίο βλέπουμε για ποιο διάστημα θεωρείται έγκυρο το πιστοποιητικό. Με τη λήξη του θα πρέπει να αντικατασταθεί με κάποιο άλλο, διαφορετικά μηνύματα προειδοποίησης θα εμφανίζονται στο φυλλομετρητή του χρήστη που θα ήθελε να προβάλει την ιστοσελίδα, και μόνο με τη συγκατάθεση του χρήστη η ιστοσελίδα θα εμφανίζεται.




**Εικόνα 34.** Το πιστοποιητικό του εξυπηρετητή.

## 6.2 Έλεγχος και Υλοποίηση Επιθέσεων

Όπως αναφέρθηκε και πιο πάνω το FlexTLS παρέχει ένα πλήθος σεναρίων έτοιμα για να εκτελεστούν. Το πρώτο αυτών είναι το σενάριο της χειραψίας (full handshake), στο οποίο δεν πραγματοποιείται επί της ουσίας κάποια επίθεση, αλλά γίνεται έλεγχος της πλήρης χειραψίας. Για την πραγματοποίηση της χειραψίας εκτελείται η εντολή:

*./FlexApps.exe –s fh - -connect 192.168.10.150:443*

, όπου το fh υλοποιεί το σενάριο της πλήρης χειραψίας, και το 192.168.10.150 είναι η IP διεύθυνση του εξυπηρετητή που φιλοξενεί την ιστοσελίδα για το πειραματικό μοντέλο που έχω κατασκευάσει. Το :443 ορίζει την πόρτα (port) στην οποία ο πελάτης θα

```
root@kali-PC: ~/mitls-flex-master/flex/FlexApps/bin/Release
File  Edit  View  Search  Terminal  Help
root@kali-PC:~# cd mitls-flex-master/flex/FlexApps/bin/Release
root@kali-PC:~/mitls-flex-master/flex/FlexApps/bin/Release# ./FlexApps.exe -s fh --connect 192.168.10.150:443
--- TCP : Connecting to 192.168.10.150:443
--- --- Done
--- # CLIENT HELLO : FlexClientHello.send
--- --- Payload : "0100004A030357347AF5E17E8E9EF4ECA7904653C90F260297F4431F2B151CD823C1D09031F5000002009C0100001F000B00020100000A000
8000600170018001 9BB8F000000170000FF01000100"
--- --- Protocol Version : TLS_1p2
--- --- Sid :
--- --- Client Random : 57347AF5E17E8E9EF4ECA7904653C90F260297F4431F2B151CD823C1D09031F5
--- --- Ciphersuites : [TLS_RSA_WITH_AES_128_GCM_SHA256]
--- --- Compressions : [NullCompression]
--- --- Extensions : [TLSExtensions+clientExtension+CE_ec_point_format;
 TLSExtensions+clientExtension+CE_ec_curves; TLSExtensions+clientExtension;
 TLSExtensions+clientExtension;
 TLSExtensions+clientExtension+CE_renegotiation_info]
--- # SERVER HELLO : FlexServerHello.receive
--- --- Protocol Version : TLS_1p2
--- --- Sid : 969C1ACF69B20D8542DE995646E6E8B571899AA10A196F207BFD66F8AEF8F39C
--- --- Server Random : 6C3A2A0BDF8FB7B4BEF51EDF60067DD99A52713AA74D3EAACDEC943893FFFD1F
--- --- Ciphersuite : TLS_RSA_WITH_AES_128_GCM_SHA256
--- --- Compression : NullCompression
--- --- Extensions : [TLSExtensions+serverExtension+SE_renegotiation_info]
--- --- Payload : 03036C3A2A0BDF8FB7B4BEF51EDF60067DD99A52713AA74D3EAACDEC943893FFFD1F20969C1ACF69B20D8542DE995646E6E8B571899AA10A19
6F207BFD66F8AEF8F39C009C000005FF01000100
--- # CERTIFICATE : FlexCertificate.receive
--- --- Payload : "0004F40004F1308204ED308203D5A003020102021053 1A2DA613E19C6B2794E15EFC21A3FF300D06092A864886F70D01010B050030818D310
B3009060355040613025553311D301B060355040A131453796D616E74656320436F72706F726174696E6E3130302E060355040B1327466F722054657374205075727
06F736573204F6E6C792E20204E6F206173737572616E6365732E312D302B06035504031324537 96D616E74656320547269616C205365637572652053657727665722
04341202D204733301E170D313630333131303030303030305A170D313630343130323335393539 5A3077310B300906035504061302435931123010060355040 80C094
4686572796E6569613112301006035504070C0946616D6167757374613122302006035504 0A0C194F70656E20556E697665727369747920 6F6204379707275733 11
C301A06035504030C137777772E746573746 66C6578746C732E6F726730820122300D06092A864886F70D010101050000382010F003082010A0282010100D2D0E9D9F
BF882CA7CBF9168420C91AF1A30BB89B6B38E67481EC48E5CB25C4418094BC289A3F15D4B9659BFC8F63B96007418B66DCC2C66E41928296882EA116991CD9206C7F
DACCF549521CAF27A8102E5AE470B41C8EFF26217F3E2602F686B531FCE41653B86D9FD4B6576B39516AE3267146BE992FC583423B431268D317FC8E63E3D1DC6352
807C51C52E5DDD3F5AE93B20C47D9839399293B6F99D56B671E15EE2D4C19D3C61E637103500BCF1C908E8D93824D469C16E4CCC9831B8E4DA4CECB734191259 90AE
FC1BD626F16FAF4FBBD46C9B25802949B0160DFD9C24D23A2C3BC0649D1CD4D5D7F414B76ABC0781425115 97A65F0D9496F176A0DC90203010001A382015C3082015
8302F0603551D110428302682137777772E746573746 66C6578746C732E6F7267820F746573746 66C6578746C732E6F726730090603551D1304023000300E0603551
D0F0101FF0404030205A0302B0603551D1F042430223020A01EA01C861A687474703A2F2F72652E73796D63622E636F6D2F72652E63726C30650603551D20045E305
C305A060A6086480186F845010715304C302306082B060105050702011617687474703 3A2F2F642E73796D63622E636F6D2F63707330250608 2B060105050702023
0190C1768747470733A2F2F642E73796D63622E636F6D2F72706130 1D0603551D250416301406082B060105050703010608 2B060105050703023 01F0603551D23041
830168014A9F04601FAAF3D8940E7162168714435 5D2AD0BE303606082B060105050701010 42A3028302606082B060105050730028 61A687474703A2F2F72652E737
96D63622E636F6D2F72652E637274300D06092A864886F70D01010B050003820101001 6578C3ADE8E8F0EA71641A4A2B3F0B78CB963B7BE72257DE63A18128B1F8AD
A4E70FC336F8B427733AEEF196CFD288293CAF62F0C41A390219BD01794B14E05B6423DD59FB2D207F762640A5A9664A3D36A4AF0EE65DC0BADEB441E64FEDAFB55C
4EC7AB2EE8D69A092AA514D101639F90E56284562040EF88C6E751AAE1C2F0BF0FA019A8F848E4932936FCF70B10E740E927DCD0D9022ECBDA62130E478CC94FEF16
8B0BCADA2DDB7B895993157B9FC34C690020775839A1229E5918E881E90D892B4A21989240A01127FD077305385624E302AE3B21619115D06C88DCC337E323C1FA9F
2F71935DCB2E5A58B08421A37267293A71816F9581BF2F11C3D1B"
--- # SERVER HELLO DONE : FlexServerHelloDone.receive
--- --- Payload :
--- # CLIENT KEY EXCHANGE : FlexClientKeyExchange.sendRSA
--- --- Payload : "1000010201000007B7C508785CA615C4169D3A9400698814AD80B1D5419F7291C95C371F6934B608CBFE543A1610BEFCB7255C4A6EEAB4CFF
AB58FEBD6928C90607B9DB4F1C3B7E35F304EA14A0268678C8A3FB5C19F93A1C608D18C0882016D6368BAA24E3264DAEA0EB827B3205B7E5BCD6C71B4DC12EE5B8AB
C19773CED7BB1057AC2CF5E9B55683840DFD3DF24F73B41F15B268BC62EECCEAE51E4C6E9CF02C8E92BA6012D3D14108BAA6CE5A646139F1DC7EF6101D73D3E53B98
0FFBC3BCFA627A11B3A0D89D5AE2E7F52B9DDD868C4BD68EE399205C9B09F0625C95A8E09AA72678D5CD23D1C2DE3C93270A910EE63183D678EAF8906F907CCD23A4
2AF25F10E55A375"
--- --- Pre Master Secret : "03039456BFD730F3A02C580D2A7202F734685010AA7876BE1BABE72EE979B6EB2389C26E49B1A5D582AB05A36644C641"
--- @ Fill Secrets
--- --- Pre Master Secret : "03039456BFD730F3A02C580D2A7202F734685010AA7876BE1BABE72EE979B6EB2389C26E49B1A5D582AB05A36644C641"
--- --- Master Secret : "170A0EA68BA608377996AB5FECA1205EFFDEA2CB0D98A3243B2CC705068F6D09D2B712156A0954C1D573D69393313871"
--- --- Reading Keys : "D8C359FACED3B1FD178A23C3C76E8F4A73528A58"
--- --- Writing Keys : "4A7B9EFCF9724613F3ABB164FA2AAA6C0336F2A2"
--- # CCS : FlexCCS.send
--- --- Payload : 01
--- @ Install Write Keys
--- # FINISHED : FlexFinished.send
--- --- Verify data : "31563783BB84CCCB07EE6A0B"
--- --- Payload : "1400000C31563783BB84CCCB07EE6A0B"
--- # CCS : FlexCCS.receive
--- --- Payload : 01
--- @ Install Read Keys
--- # FINISHED : FlexFinished.receive
--- --- Expected data : "9FE6591A4D6B2DB0DAA1BC12"
--- --- Verify data: "9FE6591A4D6B2DB0DAA1BC12"
--- --- Payload : "1400000C9FE6591A4D6B2DB0DAA1BC12"
--- # ALERT : FlexAlert.send
--- --- Description : AD_close_notify
--- --- Payload : 0100
Scenario Finished.
root@kali-PC:~/mitls-flex-master/flex/FlexApps/bin/Release#
```

**Εικόνα 35.** Η πλήρης χειραψία στο FlexTLS με αλγόριθμο RSA για την ανταλλαγή κλειδιών.

αναζητήσει την ιστοσελίδα και είναι διεθνώς ορισμένη ως το πρωτόκολλο SSL πάνω από το http, δηλαδή https. Η παράληψη του :443 δε θα επηρεάσει τη σύνδεση, μιας και ο εξυπηρετητής είναι ορισμένος να ανακατευθύνει τις μη ασφαλές συνδέσεις σε ασφαλές μέσω της πόρτας 443. Η εντολή μπορεί να πάρει ακόμα δυο παραμέτρους ως είσοδο, αυτή του ρόλου που θα επιτελέσει το FlexTLS με προεπιλογή το ρόλο του πελάτη, αν δεν οριστεί, και η άλλη είναι ο αλγόριθμος ανταλλαγής κλειδιού, με προεπιλογή το RSA.

Στην εικόνα 35 βλέπουμε την εκτέλεση της πλήρης χειραψίας. Αρχικά ο πελάτης (FlexTLS) στέλνει το FlexClientHello μήνυμα, που είναι το ίδιο με το ClientHello μήνυμα, και περιέχει τη μέγιστη έκδοση του πρωτοκόλλου που υποστηρίζει (TLS v1.2), το session_id (Sid) που είναι κενό, αφού είναι η πρώτη προσπάθεια να συνδεθεί στον εξυπηρετητή, προτείνει τους αλγόριθμους προστασίας που θέλει να χρησιμοποιηθούν, εδώ φαίνεται και η προσφορά του RSA ως αλγόριθμου ανταλλαγής κλειδιού, και τέλος ένα τυχαίο αριθμό ClientRandom. Στη συνέχεια ο εξυπηρετητής απαντά με το ServerHello μήνυμα, που περιέχει την απάντηση προς το ClientHello μήνυμα, δηλαδή το πρωτόκολλο επικοινωνίας, το session_id, τους αλγόριθμους ασφαλείας που θα χρησιμοποιηθούν και το τυχαίο αριθμό ServerRandom. Στη συνέχεια στέλνει το πιστοποιητικό του και τέλος το ServerHelloDone για να δηλώσει το τέλος της φάσης της διαπραγμάτευσης. Ακολούθως ο πελάτης στέλνει το ClientKeyExchange με το κρυπτογραφημένο με το δημόσιο RSA κλειδί του εξυπηρετητή, pre_master_secret. Βάση του pre_master_secret οι δυο πλευρές υπολογίζουν το master_secret, και ανταλλάσσουν τα κλειδιά. Στο τελευταίο μέρος ο πελάτης στέλνει το CCS μήνυμα, υποδηλώνοντας ότι απ' εδώ και πέρα τα μηνύματα θα είναι κρυπτογραφημένα, και το Finished μήνυμα για να αυθεντικοποιηθεί βάση των μηνυμάτων που έχουν σταλεί έως τώρα. Ο εξυπηρετητής στέλνει και αυτός το μήνυμα CCS, για να υποδηλώσει ότι όλα τα δεδομένα είναι κρυπτογραφημένα και το Finished μήνυμα για να αυθεντικοποιηθεί. Η χειραψία πραγματοποιήθηκε με επιτυχία, και τερματίζεται στο τελευταίο μήνυμα.

Στη συνέχεια ελέγχω την περίπτωση πλήρους χειραψίας με την προσφορά του αλγορίθμου ανταλλαγής κλειδιών του εφήμερου Diffie-Hellman. Η χειραψία ακολουθεί το πρότυπο όπως και στην προηγούμενη περίπτωση, μόνο που, όπως φαίνεται και στην εικόνα 36, ο εξυπηρετητής αυτή τη φορά δεν παρακάμπτει αλλά στέλνει το ServerKeyExchange μήνυμα, το οποίο περιέχει τις παραμέτρους του Diffie-Hellman.

```
root@kali-PC: ~/mitls-flex-master/flex/FlexApps/bin/Release
File  Edit  View  Search  Terminal  Help
4EC7AB2EE8D69A092AA514D101639F90E56284562040EF88C6E751AAE1C2F0BF0FA019A8F848E4932936FCF70B10E740E927DCD0D9022ECBDA62130E478CC94FEF16
8B0BCADA2DDB7B895993157B9FC34C690020775839A1229E5918E881E90D892B4A21989240A01127FD077305385624E302AE3B21619115D06C88DCC337E323C1FA9F
2F71935DCB2E5A58B08421A37267293A71816F9581BF2F11C3D1B"
--- # SERVER KEY EXCHANGE : FlexServerKeyExchange.receiveDHE
--- ---S Public Prime : FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A
431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3D
C2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA18217C3290
5E462E36CE3BE39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF6955817183995497CEA956AE515D2261898FA051015728E5A8AACAA68FFFFFFFF
FFFFFFFF
--- ---S Public Generator : 02
--- ---S Public Exponent : 80645256BCC6CF488C205468A0FC6A21571C27580651DEF05D917B95F473F9E03B2C48AE7BAB24798DFEB372B8899D4A476D568DD
F7E4397E94935088DF3DBFE0CF83D1E2D666AB078D9A08CE59DA26F1D15811878A617AA7768B1C7F8341A34FA3A157EC7441F04DBCC118D13C098A514724EB7F60D7
70EB2893F13FBD9C8CEE93CF3FED2400D626DE2A181501C4463921F71262FE5D20BD945260A53E0822ED06D9236D063C2EF2B6710DD772B20051A7B1B060E7BBBEBD
C27D98200F1B67C714806E5B0255A1FEA086AD69138742F4A07FEB43CBC757F2FF6DE69209BDACFB5038DAA5E0899A51F37FF697E011766AAC0D4C3095226F2D5958
F74FDCA1343
--- ---S Signature algorithm : (SA_RSA, SHA)
--- ---S Signature : A43AE08DE87B1E804374F133939D2867287BC383A218E679CA7BA8D7FC779F362B83D253CC69711D57229B16A8BA3563A55930AEC37EFF0
7E97A248478B6D3569C7E0E33AD05E0A42F199D9DDB24B7014CDD5793CC177863F9F629AFA917A4AF6EEE50BE7699507D117BB4572FF3174B916FDD67CA8A0D050C6
9FEBF6DA2FCB719C1402FE0A1F06437089217014 1E346A0CEF24876236830AEDBC7762B03022E0A847DEDAE23BB229DFCD3B8BBF01C0A54F46BA1A33F5422ACF4084
B3423D579406CFBFEE2EDC39E17F087D26639FE61251DC460819D87702222C004ECAA5913D8067DD579DB94DDBAC5DABE62981E39CAC4201529A8CAC9AEE0EBB3429
638E3
--- --- Payload : 0100FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A43
1B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2
007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA18217C32905E
462E36CE3BE39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF6955817183995497CEA956AE515D2261898FA051015728E5A8AACAA68FFFFFFFFFF
FFFFFF0001020100806452 56BCC6CF488C205468A0FC6A21571C27580651DEF05D917B95F473F9E03B2C48AE7BAB24798DFEB372B8899D4A476D568DDF7E4397E949
35088DF3DBFE0CF83D1E2D666AB078D9A08CE59DA26F1D15811878A617AA7768B1C7F8341A34FA3A157EC7441F04DBCC118D13C098A514724EB7F60D770EB2893F13
FBD9C8CEE93CF3FED2400D626DE2A181501C4463921F71262FE5D20BD945260A53E0822ED06D9236D063C2EF2B6710DD772B20051A7B1B060E7BBBEBDC27D98200F1
B67C714806E5B0255A1FEA086AD69138742F4A07FEB43CBC757F2FF6DE69209BDACFB5038DAA5E0899A51F37FF697E011766AAC0D4C3095226F2D5958F74FDCA1343
02010100A43AE08DE87B1E804374F133939D2867287BC383A218E679CA7BA8D7FC779F362B83D253CC69711D57229B16A8BA3563A55930AEC37EFF07E97A248478B6
D3569C7E0E33AD05E0A42F199D9DDB24B7014CDD5793CC177863F9F629AFA917A4AF6EEE50BE7699507D117BB4572FF3174B916FDD67CA8A0D050C69FEBF6DA2FCB7
19C1402FE0A1F06437089217014 1E346A0CEF24876236830AEDBC7762B03022E0A847DEDAE23BB229DFCD3B8BBF01C0A54F46BA1A33F5422ACF4084B3423D579406C
FBFEE2EDC39E17F087D26639FE61251DC460819D87702222C004ECAA5913D8067DD579DB94DDBAC5DABE62981E39CAC4201529A8CAC9AEE0EBB3429638E3
--- # SERVER HELLO DONE : FlexServerHelloDone.receive
--- --- Payload :
--- # CLIENT KEY EXCHANGE : FlexClientKeyExchange.sendDHE
--- --- Payload : "100001020100D3E987B4FE584ECA1C56EB7BC4BB503B15594639DCE02D1539959B066DB8E634C246990EBD4B3119BB578014FBD7EE7892241
562EF2A7495A887929F3FBB19BFF65B27D7343D6B8D8CC7EF99692933EBD9950D36B2930DA055CAA1BB50E4F9A0409B96519C3F48971E539747FB06D83E56B22EC1F
66AD27B11DA831C8B1C1F1C283E6D1F7B28D9E5A400C703EBD2ADB2A610C38135131E5E27EFFF1187B37396398A2C72959EB727BDC50439ABA7D44112C92880B7194
EE7126D502AF7408FC11C45EB67B6D1594F99746776C713292F18E17F5C156B74859BEE35629C83452D7F5CAF3190FE02CEC2DC70028FE971B81361E0CDB84B54067
9BC560F787B8CEF"
--- --- SECRET Value : 312076E513C2011262B988A49A8B5E964953C067E8DB890F2A50476D12229997578B91EEE955F1D4EE68E1FE530D813293C80B6575376
BAAEC0C2E8AB47F86784FB24EA3DF5C753E5D079A822E8F94CCB38924C12BC7A1CAE7161AFED648AAB65DD995318FA0A7948614DE8D6C2E2D0F4C68F72AE95B4D9FF
12B7EA1D1EEACA7148546BBFE4A55ADC77EDFBA69F47AB662076520CEFB434221A941D830B5EF892336448B0573EA07F165FBEFC7BC99A636AF00C205F89E8AAAFB9
B5AE01A6873946B344BC37C055041F783454CF76F98489592BA07F5FF80C680AAB207B9702C82CC4AACD2A82C11467963DCF5290D4E031695F5D5A7F1A220DC0827D
52419EE
--- --- Public Exponent : D3E987B4FE584ECA1C56EB7BC4BB503B15594639DCE02D1539959B066DB8E634C246990EBD4B3119BB578014FBD7EE7892241562EF
2A7495A887929F3FBB19BFF65B27D7343D6B8D8CC7EF99692933EBD9950D36B2930DA055CAA1BB50E4F9A0409B96519C3F48971E539747FB06D83E56B22EC1F66AD2
7B11DA831C8B1C1F1C283E6D1F7B28D9E5A400C703EBD2ADB2A610C38135131E5E27EFFF1187B37396398A2C72959EB727BDC50439ABA7D44112C92880B7194EE712
6D502AF7408FC11C45EB67B6D1594F99746776C713292F18E17F5C156B74859BEE35629C83452D7F5CAF3190FE02CEC2DC70028FE971B81361E0CDB84B540679BC56
0F787B8CEF
--- @ Fill Secrets
--- --- Pre Master Secret : "EA838FA451BF78E210F829B16E06C1BE690D31BCF354A31A18A45526AA8A015F8146A15DD7BE7125999C0551640918410A2556E
AE77C4C912C34197F43F4F4C3E2A741282BEAE133CD031107CA09337AFDCA3FD41DAC3196E46F20FBF5DA1E9C1AD9BDD289F748243BD0805F180B2F5C4B6FBC5624B
AFB229296964E0118D0569549F6A79ED06D2EF49C4014FF35F294561ECA270A5FAC92D8DDF0DDEFF074EC140AE22E64CA91BE8847C4D1E7D181D945FC7A02F10D322
2D2CC388CE1E8D6E5E39C545F9FFF83BBDBE7F223258DF35708924EF66555DA06BB87E9F5DD25A14720EAACC37C58BEE281B4C9B337FB1A827D5B2E5F25EA11877F6
503714AFFC4EB"
--- --- Master Secret : "A050B2D474190ADCCA30C8625C5EEFC25B916C39510969AA3F34963662661CABE3F6936EDEEA664970CD98C0EF61D5D0"
--- --- Reading Keys : "9AF28C730450D87249D3315AE803A8CB3DBBC920"
--- --- Writing Keys : "0EE05FA388EFD60B7C11F551354AEB2C6B293AD9"
--- # CCS : FlexCCS.send
--- --- Payload : 01
--- @ Install Write Keys
--- # FINISHED : FlexFinished.send
--- --- Verify data : "36EA8F38E2A6C2314CE2CE9C"
--- --- Payload : "1400000C36EA8F38E2A6C2314CE2CE9C"
--- # CCS : FlexCCS.receive
--- --- Payload : 01
--- @ Install Read Keys
--- # FINISHED : FlexFinished.receive
--- --- Expected data : "F1BC58993F9A569DE50019CF"
--- --- Verify data: "F1BC58993F9A569DE50019CF"
--- --- Payload : "1400000CF1BC58993F9A569DE50019CF"
Scenario Finished.
root@kali-PC:~/mitls-flex-master/flex/FlexApps/bin/Release#
```

**Εικόνα 36.** Η πλήρης χειραψία στο FlexTLS με αλγόριθμο εφήμερο Diffie-Hellman (DHE) για την ανταλλαγή κλειδιών.

Ο πελάτης ακολούθως υπολογίζει μια μυστική ποσότητα την οποία στέλνει στον εξυπηρετητή μέσω του μηνύματος ClientKeyExchange. Τέλος έχουμε τον υπολογισμό του pre_master_secret και στη συνέχεια του master_secret, και αφού ανταλλαχθούν τα CCS και Finished, το σενάριο τερματίζεται.

Αφού εξετάστηκε η περίπτωση της χειραψίας ακολούθως θα ελέγξουμε για τυχόν ευπάθειες στο σύστημά μας. Ο έλεγχος αυτός γίνεται μέσω ενός εκπληκτικού σεναρίου που αναπτύχθηκε στο FlexTLS. Το σενάριο αυτό ονομάζεται smackTLS και το οποίο στέλνει διάφορα ίχνη (deviant traces), αναμένοντας να εντοπίσει κάποια ευπάθεια. Επί της ουσίας, το smackTLS υλοποιεί διάφορα σενάρια, για να ελέγξει τυχόν αδυναμίες στον εξυπηρετητή. Το κάθε ένα από αυτά τα σενάρια είτε θα διακοπεί από το εξυπηρετητή, αντιλαμβανόμενος τις τροποποιήσεις στα μηνύματα του πρωτοκόλλου TLS, είτε θα στείλει κάποιο προειδοποιητικό μήνυμα και δε θα διακόψει τη σύνδεση, είτε θα συνεχίσει απρόσκοπτα τη σύνδεση χωρίς να στείλει κάποιο προειδοποιητικό μήνυμα. Οι δυο αυτές περιπτώσεις που η σύνδεση δεν τερματίζεται, μπορούν να σημειωθούν ως πιθανότητα ευπάθειας και να εξεταστούν περαιτέρω.

Τα σενάρια που εξετάζει το smackTLS, μπορούν να διαχωριστούν σε τρείς διαφορετικές περιπτώσεις. Πρώτη περίπτωση είναι η περίπτωση της παράληψης (skip). Η παραλήψεις μηνυμάτων υπάρχουν στο ίδιο το πρωτόκολλο, λόγω των διαφόρων αλγορίθμων ανταλλαγής κλειδιού που υποστηρίζει. Στην περίπτωση της παράληψης εξετάζουμε την παράληψη ενός ή μιας σειράς μηνυμάτων, όπως για παράδειγμα το ClientKeyExchange στην περίπτωση του Diffie-Hellman. Ακόμα μια περίπτωση είναι η μεταπήδηση καταστάσεων (hop). Η περίπτωση αυτή μοιάζει πολύ με την παράληψη μηνυμάτων μόνο που στην περίπτωση αυτή, εξετάζεται το γεγονός της μεταπήδησης από μια κατάσταση σε μια άλλη, όπως για παράδειγμα με το ClientHello (resumption), όπου ο πελάτης ζητά επανασύνδεση με το εξυπηρετητή, χωρίς όμως να έχει συνδεθεί. Τέλος είναι η περίπτωση της επανάληψης (repeat). Στην περίπτωση αυτή έχουμε επανάληψη κάποιων μηνυμάτων σε διαφορετικές στιγμές της χειραψίας, όπου και δεν αναμένονται. Για παράδειγμα η αποστολή ξανά του ClientHello στο μέσο της χειραψίας (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015, p. 5) (Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y., 2015).

```
--- BEGIN deviant trace 137
--- TCP : Connecting to 192.168.10.150:443
--- --- Done
--- # CLIENT HELLO : FlexClientHello.send
--- --- Payload : "0100004A03015734AC9ACDFE943ABC2C05DC8EA6A42F5B9035783101B161CE03C94AD0198110000000200330100001F000B00020100000A000
8000600170018001 9BB8F000000170000FF01000100"
--- --- Protocol Version : TLS_1p0
--- --- Sid :
--- --- Client Random : 5734AC9ACDFE943ABC2C05DC8EA6A42F5B9035783101B161CE03C94AD0198110
--- --- Ciphersuites : [TLS_DHE_RSA_WITH_AES_128_CBC_SHA]
--- --- Compressions : [NullCompression]
--- --- Extensions : [TLSExtensions+clientExtension+CE_ec_point_format;
 TLSExtensions+clientExtension+CE_ec_curves; TLSExtensions+clientExtension;
 TLSExtensions+clientExtension;
 TLSExtensions+clientExtension+CE_renegotiation_info]
--- # SERVER HELLO : FlexServerHello.receive
--- unsupported: Error at FlexRecord.fs:82: Error at Tcp.fs:76: TCP connection closed: Read failure..
--- END UNSUPPORTED deviant trace 137
--- BEGIN deviant trace 136
--- TCP : Connecting to 192.168.10.150:443
--- --- Done
--- # CLIENT HELLO : FlexClientHello.send
--- --- Payload : "0100004A03015734AC9D4B190E4DCB1409C2B947895D15C1FE29EDB799B8CF29A68312220300000002003301000001F000B00020100000A000
8000600170018001 9BB8F000000170000FF01000100"
--- --- Protocol Version : TLS_1p0
--- --- Sid :
--- --- Client Random : 5734AC9D4B190E4DCB1409C2B947895D15C1FE29EDB799B8CF29A68312220300
--- --- Ciphersuites : [TLS_DHE_RSA_WITH_AES_128_CBC_SHA]
--- --- Compressions : [NullCompression]
--- --- Extensions : [TLSExtensions+clientExtension+CE_ec_point_format;
 TLSExtensions+clientExtension+CE_ec_curves; TLSExtensions+clientExtension;
 TLSExtensions+clientExtension;
 TLSExtensions+clientExtension+CE_renegotiation_info]
--- # SERVER HELLO : FlexServerHello.receive
--- --- Protocol Version : TLS_1p0
--- --- Sid : 94A6F47DA0C66AA43E45983863C9333AD8D22C121248548F13F14350CA34FDEF
--- --- Server Random : 756261037116E0AB719650F92E3C55B9C4C1D3F17B56444158AD7F016A663B73
--- --- Ciphersuite : TLS_DHE_RSA_WITH_AES_128_CBC_SHA
--- --- Compression : NullCompression
--- --- Extensions : [TLSExtensions+serverExtension+SE_renegotiation_info]
--- --- Payload : 030175626103711 6E0AB719650F92E3C55B9C4C1D3F17B56444158AD7F016A663B732094A6F47DA0C66AA43E45983863C9333AD8D22C121248
548F13F14350CA34FDEF0030000005FF01000100
--- # CERTIFICATE : FlexCertificate.receive
--- --- Payload : "0004F40004F1308204ED308203D5A003020102021053 1A2DA613E19C6B2794E15EFC21A3FF300D06092A864886F70D01010B050030818D310
B30090603550406130255533 11D301B060355040A131453796D616E74656320436F72706F726174696F6E3130302E060355040B1327466F722054657374205075727
06F736573204F6E6C792E20204E6F206173737572616E6365732E312D302B06035504031324537 96D616E74656320547269616C205365637572652053657727665722
04341202D204733301E170D313630333131303030303030305A170D313630343130323335393539 5A3077310B300906035504061302435931123010606035504080C094
4686572796E65696131123010606035504070C0946616D61677573746131223020606035504 0A0C194F70656E20556E69766572736979746F662043797072757331
C301A06035504030C137777772E74657374666C6578746C732E6F726730820122300D06092A864886F70D01010105000382010F003082010A0282010100D2D0E9D9F
BF882CA7CBF9168420C91AF1A30BB89B6B38E67481EC48E5CB25C4418094BC289A3F15D4B9659BFC8F63B96007418B66DCC2C66E41928296882EA116991CD9206C7F
DACCF549521CAF27A8102E5AE470B41C8EFF26217F3E2602F686B531FCE41653B86D9FD4B6576B39516AE3267146BE992FC583423B431268D317FC8E63E3D1DC6352
807C51C52E5DDD3F5AE93B20C47D9839399293B6F99D56B671E15EE2D4C19D3C61E637103500BCF1C908E8D93824D469C16E4CCC9831B8E4DA4CECB73419125990AE
FC1BD626F16FAF4FBBD46C9B25802949B0160DFD9C24D23A2C3BC0649D1CD4D5D7F414B76ABC078142511597A65F0D9496F176A0DC90203010001A382015C3082015
8302F0603551D110428302682137777772E74657374666C6578746C732E6F7267820F74657374666C6578746C732E6F72673009060355 1D130402300030 0E0603551
D0F0101FF0404030205A0302B0603551D1F042430223020A01EA01C861A687474703A2F2F72652E73796D63622E636F6D2F72652E63726C30650603551D20045E305
C305A060A6086480186F845010715304C302306082B06010505070201161768747470733A2F2F642E73796D63622E636F6D2F63707330250608 2B060105050702023
0190C1768747470733A2F2F642E73796D63622E636F6D2F727061301D0603551D250416301406082B06010505070301060 82B06010505070302301F0603551D23041
830168014A9F04601FAAF3D8940E716216871443550D2AD0BE303606082B06010505070101042A3028302606082B06010505073002861A687474703A2F2F72652E737
96D63622E636F6D2F72652E637274300D06092A864886F70D01010B05000382010100165 78C3ADE8E8F0EA71641A4A2B3F0B78CB963B7BE72257DE63A18128B1F8AD
A4E70FC336F8B427733AEEF196CFD288293CAF62F0C41A390219BD01794B14E05B6423DD59FB2D207F762640A5A9664A3D36A4AF0EE65DC0BADEB441E64FEDAFB55C
4EC7AB2EE8D69A092AA514D101639F90E56284562040EF88C6E751AAE1C2F0BF0FA019A8F848E4932936FCF70B10E740E927DCD0D9022ECBDA62130E478CC94FEF16
8B0BCADA2DDB7B895993157B9FC34C690020775839A1229E5918E881E90D892B4A21989240A01127FD077305385624E302AE3B21619115D06C88DCC337E323C1FA9F
2F71935DCB2E5A58B08421A37267293A71816F9581BF2F11C3D1B"
--- # SERVER KEY EXCHANGE : FlexServerKeyExchange.receiveDHE
--- ---S Public Prime : FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A
431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3D
C2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA18217C3290
5E462E36CE3BE39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF695581718399549 7CEA956AE515D2261898FA051015728E5A8AACAA68FFFFFFFF
FFFFFFFF
--- ---S Public Generator : 02
--- ---S Public Exponent : B7EA37B7470F9457D3EBA11BC4F032403D6D63D1788EBF350BE84F0571C1D9FF4EC82DA62F943BA5252096BEF7AE827EB6A237163
E1A69D351BB9AF0C8BE28D01ACEC4957824D8C7890DFEF5B03FCBC598EC96063343512D443E8F7AED7010BA1799375D6E724509D87666D4134B521F32AC73442BAD2
C77DD2F8788336C82EBFB158F53E2A40291E969D2A90FFB5C8635A0801E8B7843786D30902977A5B450F3378F090A727A19872C426F5C749089E2627399D6CF6A27C
0A9D6AE50904A7810D872A857F1350F0F403AE9C4FFC5645281298E0EC8F1D9F1B23E6611888DF200B4E1B1E44E285DF6BD375E3798F95DC0E33BD7CF07BCDF331AA
1964D91234E
--- ---S Signature algorithm : (SA_RSA, MD5SHA1)
--- ---S Signature : 07B56EF1CAFADAF67EB2E771076C22A25A21CA5E0F99513FE3FD2AD76BD15BE567A34307A2BC6CE3059EBAC578892DCFC3BECD4EB7D0AD0
3043FA28811494852D1CF5E611A20715B07688146A4BF3A7B84B8D7DC4CEF278BBAF3AA74E071F74AF58194797496CAD455EB616A0681A084BEF5203524A7A60A0BE
61591239382908CBF8C4348E7AA5562A00B4A563C3956418E70100FD054C26C9AEF149864ADADC2BA67DB1D3DAA5A0E476898392157599B03A82A4EAFB9FFFC4B48F
684981CEFEE32AB160799A2832C6C8C53B2C03F10C1A8F28B3B805E905E2B5AED71C8BF85B00097137DB1DE6611372377F9C53FC949B67C4F3DB849E0A5616F0F580
90B71
--- --- Payload : 0100FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A43
1B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2
007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA18217C32905E
462E36CE3BE39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF695581718399549 7CEA956AE515D2261898FA051015728E5A8AACAA68FFFFFFFFFF
FFFFFF0001020100B7EA37B7470F9457D3EBA11BC4F032403D6D63D1788EBF350BE84F0571C1D9FF4EC82DA62F943BA5252096BEF7AE827EB6A237163E1A69D351BB
9AF0C8BE28D01ACEC4957824D8C7890DFEF5B03FCBC598EC96063343512D443E8F7AED7010BA1799375D6E724509D87666D4134B521F32AC73442BAD2C77DD2F8788
336C82EBFB158F53E2A40291E969D2A90FFB5C8635A0801E8B7843786D30902977A5B450F3378F090A727A19872C426F5C749089E2627399D6CF6A27C0A9D6AE5090
4A7810D872A857F1350F0F403AE9C4FFC5645281298E0EC8F1D9F1B23E6611888DF200B4E1B1E44E285DF6BD375E3798F95DC0E33BD7CF07BCDF331AA1964D91234E
010007B56EF1CAFADAF67EB2E771076C22A25A21CA5E0F99513FE3FD2AD76BD15BE567A34307A2BC6CE3059EBAC578892DCFC3BECD4EB7D0AD03043FA28811494852
D1CF5E611A20715B07688146A4BF3A7B84B8D7DC4CEF278BBAF3AA74E071F74AF58194797496CAD455EB616A0681A084BEF5203524A7A60A0BE61591239382908CBF
8C4348E7AA5562A00B4A563C3956418E70100FD054C26C9AEF149864ADADC2BA67DB1D3DAA5A0E476898392157599B03A82A4EAFB9FFFC4B48F684981CEFEE32AB16
0799A2832C6C8C53B2C03F10C1A8F28B3B805E905E2B5AED71C8BF85B00097137DB1DE6611372377F9C53FC949B67C4F3DB849E0A5616F0F58090B71
--- # SERVER HELLO DONE : FlexServerHelloDone.receive
--- --- Payload :
--- # CLIENT HELLO : FlexClientHello.send
--- --- Payload : "0100004A03015734AC9D4B190E4DCB1409C2B947895D15C1FE29EDB799B8CF29A68312220300000002003301000001F000B00020100000A000
8000600170018001 9BB8F000000170000FF01000100"
--- --- Protocol Version : TLS_1p0
--- --- Sid :
--- --- Client Random : 5734AC9D4B190E4DCB1409C2B947895D15C1FE29EDB799B8CF29A68312220300
--- --- Ciphersuites : [TLS_DHE_RSA_WITH_AES_128_CBC_SHA]
--- --- Compressions : [NullCompression]
--- --- Extensions : [TLSExtensions+clientExtension+CE_ec_point_format;
 TLSExtensions+clientExtension+CE_ec_curves; TLSExtensions+clientExtension;
 TLSExtensions+clientExtension;
 TLSExtensions+clientExtension+CE_renegotiation_info]
--- # ALERT : FlexAlert.receive
--- exception: Error at FlexAlert.fs:61: Unexpected handshake message: HT_server_hello with payload 0301BFBF6BDE33B8A9923F36AB595EDD
10EEB2288AC5F8F60D8B1C06DB77DD9943FC203A817278E54DC5954DF2C230103ED3BD1D4CAF6B4CAE987C7F3A6F7457FB29BB0003300005FF01000100.
--- END FAILURE deviant trace 136
--- BEGIN deviant trace 126
--- TCP : Connecting to 192.168.10.150:443
--- --- Done
--- # CLIENT HELLO : FlexClientHello.send
```

**Εικόνα 37.** Το smackTLS ανακαλύπτει ευπάθεια στην επανάληψη του ClientHello μηνύματος.

Για να ξεκινήσει το FlexTLS να ψάχνει για ίχνη στον εξυπηρετητή θα πρέπει να εκτελέσουμε το σενάριο του smackTLS με την πιο κάτω εντολή:

*./FlexApps.exe –s smacktls - -connect 192.168.10.150:443*

Τότε το ξεκινά να τρέχει το ένα σενάριο μετά το άλλο κλείνοντας επιτυχώς ή ανεπιτυχώς τη σύνδεση. Στην εικόνα 37 απεικονίζεται η επανάληψη του ClientHello μηνύματος στο τέλος της χειραψίας. Ο εξυπηρετητής ανταποκρίνεται με το ServerHello μήνυμα, γι' αυτό έχουμε και το μήνυμα ειδοποίησης από το FlexTLS ότι λήφθηκε μη αναμενόμενο μήνυμα, και ακολούθως τερματίζει τη σύνδεση. Επιπλέον σε διαφορετικά ίχνη (deviant traces) που εκτελέστηκαν, ανακαλύφθηκε ευπάθεια όσον αφορά την παράληψη σειράς μηνυμάτων και την αποστολή του CCS, που οδηγούν στην επίθεση του EarlyCCS. Ελέγχθηκε όμως και οι περιπτώσεις των κατακερματισμένων μηνυμάτων, ClientHello και προειδοποίησης (alert), που δεν καλύπτονταν από τον έλεγχο των ιχνών, χειροκίνητα. Έτσι ανακαλύφθηκε ευπάθεια στην περίπτωση του κατακερματισμένου μηνύματος προειδοποίησης.

Από την άλλη υπάρχουν περιπτώσεις, όπου ο εξυπηρετητής δεν αποδέχεται τις αλλαγές στα μηνύματα του πρωτοκόλλου και τερματίζει την σύνδεση λέγοντας ότι είναι δε μπορεί να την υποστηρίξει. Στην εικόνα 37 βλέπουμε ότι το ίχνος 126 ξεκινά μετά το τέλος του 136. Στην εικόνα 38 απεικονίζεται το μήνυμα ότι δεν υποστηρίζεται το ίχνος αυτό , τερματισμός της σύνδεσης και ξεκινά καινούργιο ίχνος.

Η χρήση του smackTLS είναι ιδιαίτερα χρήσιμη για τη σκιαγράφηση και εντοπισμό πιθανών ευπαθειών. Με τον τρόπο αυτό μπορείς να προσδιορίσεις τις πιθανές επιθέσεις και να διορθώσεις τα κενά αυτά πριν είναι αργά. Η σημαντικότητα του επικυρώνεται με το γεγονός της ανάπτυξής του στον ισότοπο http://smacktest.com/, όπου ο κάθε ένας μπορεί να ελέγξει τον εξυπηρετητή του, αλλά και ο απλός χρήστης έχει τη δυνατότητα να ελέγξει το φυλλομετρητή του χωρίς την εγκατάσταση του FlexTLS ή κάποιου άλλου λογισμικού.

```
--- END UNSUPPORTED deviant trace 126
--- BEGIN deviant trace 125
--- TCP : Connecting to 192.168.10.150:443
```

**Εικόνα 368** Το τέλος του μη υποστηριζόμενου ίχνους 126.

Πρώτη απόπειρα επίθεσης είναι αυτή του κατακερματισμένου μηνύματος προειδοποίησης. Όπως, προαναφέρθηκε, το μήνυμα προειδοποίησης αποτελείτε από δυο bytes. Η τεχνική της επίθεσης βασίζεται στο κατακερματισμό ενός μηνύματος προειδοποίησης και την αποστολή μόνο του ενός byte. Επομένως, στη συνέχεια, οποιοδήποτε κρίσιμο μήνυμα προειδοποίησης και αν σταλεί, η μηχανή θα αναγνωρίσει το πρώτο byte που έλαβε, μαζί με το πρώτο μέρος της προειδοποίησης που έλαβε, αγνοώντας το δεύτερο byte. Αυτό θα υποβιβάσει το μήνυμα από κρίσιμο σε απλό προειδοποιητικό και η σύνδεση θα συνεχιστεί.

Η επίθεση εκτελείται με την εντολή:

*./FlexApps.exe –s fal - -connect 192.168.10.150:443*

και το αποτέλεσμά της απεικονίζεται στην εικόνα 39. Το πρώτο byte μηνύματος προειδοποίησης στέλνεται με την λήψη του ServerHello μηνύματος. Ακολούθως και αφού τελειώσει η χειραψία στέλνεται ένα ολόκληρο κρίσιμο μήνυμα προειδοποίησης. Όπως παρατηρούμε στην εικόνα, η σύνδεση παραμένει ενεργή μετά από αυτό, ενώ θα έπρεπε να τερματιστεί αμέσως.

Επόμενη απόπειρα είναι η επίθεση EarlyCCS, με την αποστολή του μηνύματος CCS αμέσως το ServerHelloDone μήνυμα. Η επίθεση υλοποιείται εκτελώντας την εντολή:

*./FlexApps.exe –s eccs - -connect 192.168.10.150:443*

και το αποτέλεσμα αυτής φαίνεται στην εικόνα 40. Στην εικόνα 40 βλέπουμε την αρχή της εκτέλεσης, όπου ο πελάτης στέλνει το ClientHello μήνυμα. Στη συνέχεια ακολουθούν τα μηνύματα από το εξυπηρετητή ServerHello, πιστοποιητικό και το ServerHelloDone μήνυμα, ο επιτιθέμενος στέλνει CCS μήνυμα. Αφού δεν ανταλλάχθηκαν καθόλου κρυπτογραφικές παραμέτροι, ούτε το ClientKeyExchange, το pre_master_secret είναι κενό. Έτσι το master_secret που υπολογίζεται είναι 48-byte μηδενικά. Έτσι ο επιτιθέμενος είναι σε θέση να υπολογίσει τα κλειδιά και να πάρει τον έλεγχο της σύνδεσης.

```
root@kali-PC: ~/mitls-flex-master/flex/FlexApps/bin/Release
File  Edit  View  Search  Terminal  Help
root@kali-PC:~/mitls-flex-master/flex/FlexApps/bin/Release# ./FlexApps.exe -s fal --connect 192.168.10.150
--- TCP : Connecting to 192.168.10.150:443
--- --- Done
--- # CLIENT HELLO : FlexClientHello.send
--- --- Payload : "0100005A030357363DA00195E08FE1A21FB479B82C3A7C24CAAB2688F83B9B2BB6C4E30F569D000012C02FC02F009C009E00A2002F0033003
2000A0100001F000B00020100000A00080006001700180019BB8F000000170000FF01000100"
--- --- Protocol Version : TLS_1p2
--- --- Sid :
--- --- Client Random : 57363DA00195E08FE1A21FB479B82C3A7C24CAAB2688F83B9B2BB6C4E30F569D
--- --- Ciphersuites : [TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256; TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256;
 TLS_RSA_WITH_AES_128_GCM_SHA256; TLS_DHE_RSA_WITH_AES_128_GCM_SHA256;
 TLS_DHE_DSS_WITH_AES_128_GCM_SHA256; TLS_RSA_WITH_AES_128_CBC_SHA;
 TLS_DHE_RSA_WITH_AES_128_CBC_SHA; TLS_DHE_DSS_WITH_AES_128_CBC_SHA;
 TLS_RSA_WITH_3DES_EDE_CBC_SHA]
--- --- Compressions : [NullCompression]
--- --- Extensions : [TLSExtensions+clientExtension+CE_ec_point_format;
 TLSExtensions+clientExtension+CE_ec_curves; TLSExtensions+clientExtension;
 TLSExtensions+clientExtension;
 TLSExtensions+clientExtension+CE_renegotiation_info]
--- # SERVER HELLO : FlexServerHello.receive
--- --- Protocol Version : TLS_1p0
--- --- Sid : CB452F93CC94F50467A7C6CED6252119E72850957BD9670B6C4195EF35624598
--- --- Server Random : 1E710F479796E04AFA3A9BBA07D738C9F30E47E7C3033D25A7626711D210A39F
--- --- Ciphersuite : TLS_RSA_WITH_AES_128_CBC_SHA
--- --- Compression : NullCompression
--- --- Extensions : [TLSExtensions+serverExtension+SE_renegotiation_info]
--- --- Payload : 03011E710F479796E04AFA3A9BBA07D738C9F30E47E7C3033D25A7626711D210A39F20CB452F93CC94F50467A7C6CED6252119E72850957BD9
670B6C4195EF35624598002F000005FF01000100
--- # ALERT : FlexAlert.send
--- --- Payload : 01
--- # CERTIFICATE : FlexCertificate.receive
--- --- Payload : "0004F40004F1308204ED308203D5A003020102021053 1A2DA613E19C6B2794E15EFC21A3FF300D06092A864886F70D01010B050030818D310
B3009060355040613025553311D301B060355040A131453796D616E74656320436F72706F726174696F6E3130302E060355040B1327466F722054657374205075727
06F736573204F6E6C792E20204E6F206173737572616E6365732E312D302B06035504031324537 96D616E74656320547269616C205365637572652053657727665722
04341202D204733301E170D3136303331313030303030305A170D3136303431303233353935395A3077310B3009060355040613024359311230100603550408 0C094
4686572796E65696131123010060355040 70C0946616D6167757374613122302006035504 0A0C194F70656E20556E69766572273697479206F66204379707273733 11
C301A06035504030C137777772E74657374666C6578746C732E6F72673082012230 0D06092A864886F70D01010105000382010F003082010A0282010100D2D0E9D9F
BF882CA7CBF9168420C91AF1A30BB89B6B38E67481EC48E5CB25C4418094BC289A3F15D4B9659BFC8F63B96007418B66DCC2C66E41928296882EA116991CD9206C7F
DACCF549521CAF27A8102E5AE470B41C8EFF26217F3E2602F686B531FCE41653B86D9FD4B6576B39516AE3267146BE992FC583423B431268D317FC8E63E3D1DC6352
807C51C52E5DDD3F5AE93B20C47D9839399293B6F99D56B671E15EE2D4C19D3C61E637103500BCF1C908E8D93824D469C16E4CCC9831B8E4DA4CECB73419125990AE
FC1BD626F16FAF4FBBD46C9B25802949B0160DFD9C24D23A2C3BC0649D1CD4D5D7F414B76ABC078142511597A65F0D9496F176A0DC90203010001A382015C3082015
8302F0603551D110428302682137777772E74657374666C6578746C732E6F7267820F74657374666C6578746C732E6F72673009060355 1D1304023000300E0603551
D0F0101FF0404030205A0302B0603551D1F042430223020A01EA01C861A687474703A2F2F72652E73796D63622E636F6D2F72652E63726C30650603551D20045E305
C305A060A6086480186F845010715304C302306082B06010505070201161768747470733A2F2F642E73796D63622E636F6D2F637073302506082B06010505070202 3
0190C1768747470733A2F2F642E73796D63622E636F6D2F727061301D0603551D250416301406082B06010505070301 06082B06010505070302301F0603551D23041
830168014A9F04601FAAF3D8940E7162168714435 5D2AD0BE303606082B06010505070101042A3028302606082B06010505073002861A687474703A2F2F72652E737
96D63622E636F6D2F72652E637274300D06092A864886F70D01010B05000382010100165 78C3ADE8E8F0EA71641A4A2B3F0B78CB963B7BE72257DE63A18128B1F8AD
A4E70FC336F8B427733AEEF196CFD288293CAF62F0C41A390219BD01794B14E05B6423DD59FB2D207F762640A5A9664A3D36A4AF0EE65DC0BADEB441E64FEDAFB55C
4EC7AB2EE8D69A092AA514D101639F90E56284562040EF88C6E751AAE1C2F0BF0FA019A8F848E4932936FCF70B10E740E927DCD0D9022ECBDA62130E478CC94FEF16
8B0BCADA2DDB7B895993157B9FC34C690020775839A1229E5918E881E90D892B4A21989240A01127FD077305385624E302AE3B21619115D06C88DCC337E323C1FA9F
2F71935DCB2E5A58B08421A37267293A71816F9581BF2F11C3D1B"
--- # SERVER HELLO DONE : FlexServerHelloDone.receive
--- --- Payload :
--- # CLIENT KEY EXCHANGE : FlexClientKeyExchange.sendRSA
--- --- Payload : "100001020100884 9A00D27F2320D5034FCB15710E84C7EEDCDB3AB669CF0A0B95F7A6CE2B800F3B3CC0BE8211129B7B322BB49B35407B6FFA
7CF71FC4D3BBDD4575821AFCF1F0EAB57072ED4EC3B0BF8B70C19B000AEDBB7CE802E3B1FF862C577E1A14886B68A9E753B484C84A977205EBA70EC1103D8E5C79C9
DEC1CF1F93EDA30F6651346C9909649E0316A7EF4D6FB24A136BEC193C3D399235B53BA65C398D414824B821397056FF92A807D4F09E4A698B6CAB7862316B2AC5A2
F49A211B71136358B2AB70666F62A952F58557254995C8F16B34C26D6423B9A4C7A34CF7687FE65869E14EFB3D6B95BB82394B7C8A958E310BF095850F36093FA5E9
77FF9A281E6BCBA"
--- --- Pre Master Secret : "03038B2829A69BD7576FE9E219B360FEBF70136D3048053520B96E82B99344A35091C54C7B40BBC5FEFD3AF539CB5E7F"
--- @ Fill Secrets
--- --- Pre Master Secret : "03038B2829A69BD7576FE9E219B360FEBF70136D3048053520B96E82B99344A35091C54C7B40BBC5FEFD3AF539CB5E7F"
--- --- Master Secret : "10AD3EF69E5E4EEF16E0337C60D7101E836F10CD6C627F40040BC3724F26061668AD4AB4B3C7C3C4CF145AAA98A3ACA3"
--- --- Reading Keys : "40842F0F3D440BC654D9C105E354AA93E4A5139C3C072DD5D91B51BC363B8DD323281DB2023D6AB879E036296DFA2EEB0F9B4201"
--- --- Writing Keys : "C69C3A6DED09F2D3308CD92D3ECCFBA6A620435ACF65A6AE4D41D0273A167DDC6E1A8B2D1F074383B52B88768F191EE92F6D97BF"
--- # CCS : FlexCCS.send
--- --- Payload : 01
--- @ Install Write Keys
--- # FINISHED : FlexFinished.send
--- --- Verify data : "9F1D2F3AFB96E0CC89FDCA82"
--- --- Payload : "1400000C9F1D2F3AFB96E0CC89FDCA82"
--- # CCS : FlexCCS.receive
--- --- Payload : 01
--- @ Install Read Keys
--- # FINISHED : FlexFinished.receive
--- --- Expected data : "031551CE1612D37F675A01BF"
--- --- Verify data: "031551CE1612D37F675A01BF"
--- --- Payload : "1400000C031551CE1612D37F675A01BF"
--- # APPLICATION DATA : FlexAppData.send
---> GET / HTTP/1.1
Host: 192.168.10.150
Connection: keep-alive
Cache-Control: max-age=0

--- # APPLICATION DATA : FlexAppData.receive
<--- HTTP/1.1 200 OK
Date: Fri, 13 May 2016 17:48:42 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Tue, 01 Mar 2016 18:19:06 GMT
ETag: "2d12-52d00cd009f27"
Accept-Ranges: bytes
Content-Length: 11538
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

**Εικόνα 39.** Επίθεση κατακερματισμένου μηνύματος ειδοποίησης (fragmented alert message).

```
root@kali-PC: ~/mitls-flex-master/flex/FlexApps/bin/Release
File  Edit  View  Search  Terminal  Help
root@kali-PC:~/mitls-flex-master/flex/FlexApps/bin/Release# ./FlexApps.exe -s eccs --connect 192.168.10.150:443
--- TCP : Connecting to 192.168.10.150:443
--- --- Done
--- # CLIENT HELLO : FlexClientHello.send
--- --- Payload : "0100004A0303573521A89BC843CF236B0139CCDA7DAEDE9ADBBED59E29D11D62F9D3C1EC1144000002002F0100001F000B00020100000A000
800060017001800 19BB8F000000170000FF01000100"
--- --- Protocol Version : TLS_1p2
--- --- Sid :
--- --- Client Random : 573521A89BC843CF236B0139CCDA7DAEDE9ADBBED59E29D11D62F9D3C1EC1144
--- --- Ciphersuites : [TLS_RSA_WITH_AES_128_CBC_SHA]
--- --- Compressions : [NullCompression]
--- --- Extensions : [TLSExtensions+clientExtension+CE_ec_point_format;
 TLSExtensions+clientExtension+CE_ec_curves; TLSExtensions+clientExtension;
 TLSExtensions+clientExtension;
 TLSExtensions+clientExtension+CE_renegotiation_info]
--- # SERVER HELLO : FlexServerHello.receive
--- --- Protocol Version : TLS_1p2
--- --- Sid : 533C94BC4DCF4C5D81CC676F9DEEA6FFF5D4170A4E19641C8C07D5F8EC9AB1F2
--- --- Server Random : DB17DB6173F8E80A39E3898CD66B981CBB0ECB2D1CE7A5CF851A607C7ACEDFC6
--- --- Ciphersuite : TLS_RSA_WITH_AES_128_CBC_SHA
--- --- Compression : NullCompression
--- --- Extensions : [TLSExtensions+serverExtension+SE_renegotiation_info]
--- --- Payload : 0303DB17DB6173F8E80A39E3898CD66B981CBB0ECB2D1CE7A5CF851A607C7ACEDFC620533C94BC4DCF4C5D81CC676F9DEEA6FFF5D4170A4E19
641C8C07D5F8EC9AB1F2002F000005FF01000100
--- # CERTIFICATE : FlexCertificate.receive
--- --- Payload : "0004F40004F1308204ED308203D5A003020102021053 1A2DA613E19C6B2794E15EFC21A3FF300D06092A864886F70D01010B050030818D310
B3009060355040613025553311D301B060355040A131453796D616E74656320436F72706F726174696F6E3130302E060355040B1327466F722054657374205075727
06F736573204F6E6C792E20204E6F206173737572616E6365732E312D302B06035540313245379 6D616E74656320547269616C205365637572652053657265727665722
04341202D204733301E170D313630333131303030303030305A170D313630343130323335393535A3077310B300906035504061302435931123010060355040 80C094
4686572796E65696131123010060355040 70C0946616D6167757374613122302006035504 0A0C194F70656E20556E697665727369747920 6F662043797070727573311
C301A06035504030C137777772E74657374666C6578746C732E6F726730820122300D06092A864886F70D01010105000382010F003082010A0282010100D2D0E9D9F
BF882CA7CBF9168420C91AF1A30BB89B6B38E67481EC48E5CB25C4418094BC289A3F15D4B9659BFC8F63B96007418B66DCC2C66E41928296882EA116991CD9206C7F
DACCF549521CAF27A8102E5AE470B41C8EFF26217F3E2602F686B531FCE41653B86D9FD4B6576B39516AE3267146BE992FC583423B431268D317FC8E63E3D1DC6352
807C51C52E5DDD3F5AE93B20C47D9839399293B6F99D56B671E15EE2D4C19D3C61E637103500BCF1C908E8D93824D469C16E4CCC9831B8E4DA4CECB73419125990AE
FC1BD626F16FAF4FBBD46C9B25802949B0160DFD9C24D23A2C3BC0649D1CD4D5D7F414B76ABC078142511597A65F0D9496F176A0DC90203010001A382015C3082015
8302F0603551D110428302682137777772E74657374666C6578746C732E6F7267820F74657374666C6578746C732E6F726730090603551D1304023000300E0603551
D0F0101FF0404030205A0302B0603551D1F042430223020A01EA01C861A687474703A2F2F72652E73796D63622E636F6D2F72652E63726C30650603551D20045E305
C305A060A6086480186F845010715304C302306082B060105050702011617687474707 33A2F2F642E73796D63622E636F6D2F637073302506082B060105050702023
0190C1768747470733A2F2F642E73796D63622E636F6D2F727061301D0603551D250416301406082B060105050703010608 2B060105050703020301F0603551D23041
830168014A9F04601FAAF3D8940E71621687144355D2AD0BE303606082B060105050701010 42A3028302606082B060105050730028 61A687474703A2F2F72652E737
96D63622E636F6D2F72652E637274300D06092A864886F70D01010B050003820101001 6578C3ADE8E8F0EA71641A4A2B3F0B78CB963B7BE72257DE63A18128B1F8AD
A4E70FC336F8B427733AEEF196CFD288293CAF62F0C41A390219BD01794B14E05B6423DD59FB2D207F762640A5A9664A3D36A4AF0EE65DC0BADEB441E64FEDAFB55C
4EC7AB2EE8D69A092AA514D101639F90E56284562040EF88C6E751AAE1C2F0BF0FA019A8F848E4932936FCF70B10E740E927DCD0D9022ECBDA62130E478CC94FEF16
8B0BCADA2DDB7B895993157B9FC34C690020775839A1229E5918E881E90D892B4A21989240A01127FD077305385624E302AE3B21619115D06C88DCC337E323C1FA9F
2F71935DCB2E5A58B08421A37267293A71816F9581BF2F11C3D1B"
--- # SERVER HELLO DONE : FlexServerHelloDone.receive
--- --- Payload :
--- # CCS : FlexCCS.send
--- --- Payload : 01
--- @ Fill Secrets
--- --- Pre Master Secret : ""
--- --- Master Secret : "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
--- --- Reading Keys : "863DEDC5518E4D4533BE71B756225A081839C20A275194BC5DF012F2441B946470AEC30B"
--- --- Writing Keys : "9610B299D17DDBC709A37AA57B5AE6AD286B4EEAE0C6F3D9CC4FB46AD0D57F6EE5CBF7F9"
--- @ Install Write Keys
```

**Εικόνα 40.** Η επίθεση EarlyCCS.

Η επόμενη επίθεση έχει να κάνει με την αποδοχή δεδομένων κατά τη διαδικασία του πρωτοκόλλου χειραψίας Early AppData. Εκτελείται με την εντολή:

*./FlexApps.exe –s eapp - -connect 192.168.10.150:443*

και το αποτέλεσμά της φαίνεται στις εικόνες 41. Όπως και στην προηγούμενη επίθεση έτσι και σ' αυτή στην εικόνα 41 φαίνεται το ClientHello μήνυμα από το πελάτη. Στη συνέχει ο εξυπηρετητής απαντά κανονικά με το ServerHello, το πιστοποιητικό και το ServerHelloDone. Αμέσως μετά ο επιτιθέμενος ξεκινά να στέλνει δεδομένα σε μορφή απλού κειμένου. Ο εξυπηρετητής απαντά στον επιτιθέμενο με δεδομένα απλού

κειμένου. Η επίθεση αυτή είναι σε πειραματική μορφή στο FlexTLS, γι' αυτό και βρισκόταν στην κατηγορία Tests.

```
root@kali-PC: ~/mitls-flex-master/flex/FlexApps/bin/Release
File  Edit  View  Search  Terminal  Help
root@kali-PC:~/mitls-flex-master/flex/FlexApps/bin/Release# ./FlexApps.exe -s eapp --connect 192.168.10.150:443
--- TCP : Connecting to 192.168.10.150:443
--- --- Done
--- # CLIENT HELLO : FlexClientHello.send
--- --- Payload : "0100004A030357351D62227D0CF9CEDACE0DA4F23C09D98C30468CD6176E6635F46F88A04FC3000002009C0100001F000B00020100000A000
8000600170018001 9BB8F000000170000FF01000100"
--- --- Protocol Version : TLS_1p2
--- --- Sid :
--- --- Client Random : 57351D62227D0CF9CEDACE0DA4F23C09D98C30468CD6176E6635F46F88A04FC3
--- --- Ciphersuites : [TLS_RSA_WITH_AES_128_GCM_SHA256]      53B73  53B73
--- --- Compressions : [NullCompression]
--- --- Extensions : [TLSExtensions+clientExtension+CE_ec_point_format;
 TLSExtensions+clientExtension+CE_ec_curves; TLSExtensions+clientExtension;
 TLSExtensions+clientExtension;
 TLSExtensions+clientExtension+CE_renegotiation_info]
--- # SERVER HELLO : FlexServerHello.receive
--- --- Protocol Version : TLS_1p2
--- --- Sid : E4B94EDD36BE3A0F1584FC7918E8D14AF4A0A97F6C40E820DE89FAD063DA93FB
--- --- Server Random : D55A846B06C26A32BF849B8E581352C74784E709606B8A0467C2BE5236B2E621
--- --- Ciphersuite : TLS_RSA_WITH_AES_128_GCM_SHA256
--- --- Compression : NullCompression
--- --- Extensions : [TLSExtensions+serverExtension+SE_renegotiation_info]
--- --- Payload : 0303D55A846B06C26A32BF849B8E581352C74784E709606B8A0467C2BE5236B2E62120E4B94EDD36BE3A0F1584FC7918E8D14AF4A0A97F6C40
E820DE89FAD063DA93FB009C000005FF01000100
--- # CERTIFICATE : FlexCertificate.receive
--- --- Payload : "0004F40004F1308204ED308203D5A003020102021053 1A2DA613E19C6B2794E15EFC21A3FF300D06092A864886F70D01010B050030818D310
B3009060355040613025553311D301B060355040A131453796D616E74656320436F72706F726174696F6E3130302E060355040B1327466F722054657374205075727
06F736573204F6E6C792E20204E6F206173737572616E6365732E312D302B06035504031324537 96D616E74656320547269616C205365637572652053657276657722
04341202D204733301E170D3136303331313030303030305A170D3136303431303233353935395A3077310B3009060355040613024359311230100603550408 0C094
4686572796E65696131123010060355040 70C0946616D6167757374613122302006035504 0A0C194F70656E20556E69766572736974 79206F662043797072757331 1
C301A06035504030C137777772E74657374666C6578746C732E6F72673082012230 0D06092A864886F70D01010105000382010F003082010A0282010100D2D0E9D9F
BF882CA7CBF9168420C91AF1A30BB89B6B38E67481EC48E5CB25C4418094BC289A3F15D4B9659BFC8F63B96007418B66DCC2C66E41928296882EA116991CD9206C7F
DACCF549521CAF27A8102E5AE470B41C8EFF26217F3E2602F686B531FCE41653B86D9FD4B6576B39516AE3267146BE992FC583423B431268D317FC8E63E3D1DC6352
807C51C52E5DDD3F5AE93B20C47D9839399293B6F99D56B671E15EE2D4C19D3C61E637103500BCF1C908E8D93824D469C16E4CCC9831B8E4DA4CECB73419125990AE
FC1BD626F16FAF4FBBD46C9B25802949B0160DFD9C24D23A2C3BC0649D1CD4D5D7F414B76ABC078142511597A65F0D9496F176A0DC90203010001A382015C3082015
8302F0603551D110428302682137777772E74657374666C6578746C732E6F7267820F74657374666C6578746C732E6F72673009060355 1D1304023000300E0603551
D0F0101FF0404030205A0302B0603551D1F042430223020A01EA01C861A687474703A2F2F72652E73796D63622E636F6D2F72652E63726C30650603551D20045E305
C305A060A6086480186F845010715304C302306082B06010505070201161768747470733A2F2F642E73796D63622E636F6D2F637073302506082B06010505070202 3
0190C1768747470733A2F2F642E73796D63622E636F6D2F727061301D0603551D250416301406082B06010505070301 06082B06010505070302301F0603551D23041
830168014A9F04601FAAF3D8940E71621687144355D2AD0BE303606082B06010505070101042A3028302606082B06010505073002861A687474703A2F2F72652E737
96D63622E636F6D2F72652E637274300D06092A864886F70D01010B05000382010100 16578C3ADE8E8F0EA71641A4A2B3F0B78CB963B7BE72257DE63A18128B1F8AD
A4E70FC336F8B427733AEEF196CFD288293CAF62F0C41A390219BD01794B14E05B6423DD59FB2D207F762640A5A9664A3D36A4AF0EE65DC0BADEB441E64FEDAFB55C
4EC7AB2EE8D69A092AA514D101639F90E56284562040EF88C6E751AAE1C2F0BF0FA019A8F848E4932936FCF70B10E740E927DCD0D9022ECBDA62130E478CC94FEF16
8B0BCADA2DDB7B895993157B9FC34C690020775839A1229E5918E881E90D892B4A21989240A01127FD077305385624E302AE3B21619115D06C88DCC337E323C1FA9F
2F71935DCB2E5A58B08421A37267293A71816F9581BF2F11C3D1B"
--- # SERVER HELLO DONE : FlexServerHelloDone.receive
--- --- Payload :
--- # APPLICATION DATA : FlexAppData.send
--- # APPLICATION DATA : FlexAppData.receive
```

**Εικόνα 41.** Η επίθεση Early AppData.

## 6.3 Σύγκριση Αποτελεσμάτων και Εξαγωγή Συμπερασμάτων

Οι επιθέσεις πραγματοποιήθηκαν με επιτυχία, επιβεβαιώνοντας δυο πράγματα. Πρώτον την χρησιμότητα του εργαλείου FlexTLS, ως εργαλείο εντοπισμού ευπαθειών στο

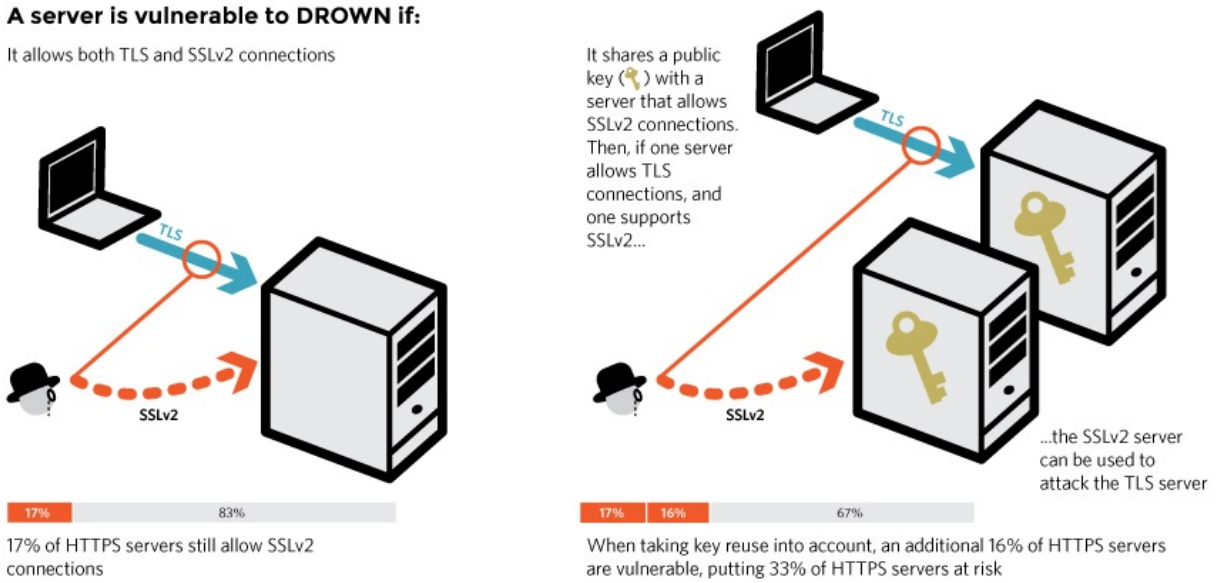
πρωτόκολλο TLS. Και δεύτερον, τις ευπάθειες που φέρουν παλιότερες εκδόσεις του OpenSSL, όπως η έκδοση 1.0.1f που χρησιμοποιήθηκε στον Apache εξυπηρετητή.

Οι επιθέσεις που υλοποιήθηκαν και αναλύθηκαν στην παρούσα Μεταπτυχιακή Διατριβή, μπορεί να μην θεωρούνται ως οι μεγάλες αυτές επιθέσεις που μπορούν να οδηγήσουν στην κατάρρευση του πρωτοκόλλου TLS v1.2 και στην αναθεώρησή του, όπως είναι οι επιθέσεις του FREAK και Logjam, δε παύουν όμως να αποτελούν ευπάθειες του πρωτοκόλλου που πρέπει να διορθωθούν. Ενώ η έκδοση του OpenSSL ήταν επιρρεπής στις σημαντικές αυτές επιθέσεις του FREAK και του Logjam, η πραγματοποίηση τους δεν ήταν δυνατή λόγω της έλλειψης των αλγορίθμων που μπορούσαν να προϋπολογίσουν τις παραμέτρους των κλειδιών RSA και DH, και επιπλέον της έλλειψης τέτοιας υπολογιστικής δύναμης που χρησιμοποιήθηκε στις δημοσιευμένες έρευνες (Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, A., J., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E, Zanella-Beguelin, S., Zimmermann P., 2015) (Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K., 2015).

Ένα χρόνο μετά τις μεγάλες αυτές επιθέσεις, και ενώ η Μεταπτυχιακή Διατριβή βρισκόταν σε εξέλιξη, το Μάρτιο του 2016, μια νέα επίθεση είδε το φως της δημοσιότητας. Η επίθεση αυτή αναφέρεται ως DROWN (Decrypting RSA using Obsolete and Weakened eNcryption) και αναφέρεται στο Common Vulnerabilities and Exposures (CVE-2016-0800). Η επίθεση εντάσσεται και αυτή ανάμεσα στις επιθέσεις διασταυρωμένου πρωτοκόλλου, μιας και υποβαθμίζει τη σύνδεση στο απαρχαιωμένο και εξαιρετικά επιρρεπή πρωτόκολλο του SSL v2.0.

Το SSL v2.0 αποσύρθηκε, περίπου, ένα χρόνο μετά τη δημοσίευσή του γιατί τόσο αυτό, όσο και το προηγούμενο SSL v1.0, ήταν εξαιρετικά επιρρεπή και γεμάτα κενά ασφαλείας. Το αμέσως επόμενο πρωτόκολλο SSL v3.0 είχε ένα εντελώς ανασχεδιασμένο και διαφορετικό πρωτόκολλο χειραψίας, που περιγράφεται στο κεφάλαιο 3.1.4. Το πρωτόκολλο χειραψίας του SSL v2.0 ξεκινούσε με το ClientHello μήνυμα που περιλάμβανε μια λίστα από τους υποστηριζόμενους από τον πελάτη αλγορίθμους κρυπτογράφησης και ένα τυχαίο νούμερο. Ο εξυπηρετητής απαντούσε με ένα ServerHello μήνυμα που περιελάβανε μια λίστα με τους υποστηριζόμενους από αυτό

αλγορίθμους κρυπτογράφησης και ένα τυχαίο αριθμό. Στη συνέχεια ο πελάτης έστελνε το ClientMasterKey που περιλάμβανε τον επιλεγμένο αλγόριθμο κρυπτογράφησης και το master_secret. Το master_secret χωριζόταν σε δυο μέρη, ένα κρυπτογραφημένο και ένα απλό κείμενο. Ακολούθως πελάτης και εξυπηρετητής υπολόγιζαν από το πιο πάνω συνδυασμό το τελικό master_secret. Ο πελάτης έστελνε το ClientFinished αυθεντικοποιόντας τον εαυτό του. Τέλος ο εξυπηρετητής έστελνε ένα ServerVerify και ένα ServerFinished για αυθεντικοποίηση. Παρόλα αυτά, πολλοί εξυπηρετητές, μέχρι το Μάρτιο του 2016, υποστήριζαν το πρωτόκολλο SSL v2.0, θεωρώντας το ασφαλές, αφού κανένας φυλλομετρητής δε πρόσφερε κατά τη χειραψία, το πρωτόκολλο αυτό. Ακόμα πιο ανησυχητικό είναι το γεγονός, ότι προηγούμενες εκδόσεις, 0.9.8 και 1.0.0, της κρυπτογραφικής βιβλιοθήκης OpenSSL υποστήριζαν το SSL v2.0.




**Εικόνα 42.** Πότε ένας εξυπηρετητής είναι επιρρεπής στην επίθεση DROWN (Aviram, N., Schinzel, S., Somorovsky, J., Heninger, N., Dankel, M., Steube, J., Valenta, L., Adrian, D., Halderman, J., A., Dukhovni, V., Käsper, E., Cohney, S., Engels, S., Paar, C., Shavitt, Y., 2016).

Ένας εξυπηρετητής είναι επιρρεπείς στην επίθεση DROWN σε δυο περιπτώσεις όπως φαίνεται και στην εικόνα 42. Στην πρώτη περίπτωση ο ίδιος ο εξυπηρετητής υποστηρίζει το πρωτόκολλο SSL v2.0. Στη δεύτερη περίπτωση διάφοροι εξυπηρετητές χρησιμοποιούν το ίδιο δημόσιο κλειδί, αν ο ένας υποστηρίζει το SSL v2.0, τότε όλοι οι υπόλοιποι εξυπηρετητές θεωρούνται επιρρεπείς στο DROWN. Αυτό συμβαίνει γιατί ο επιτιθέμενος θα αποκτήσει το ιδιωτικό κλειδί, από τον εξυπηρετητή που υποστηρίζει SSL v2.0, που είναι κοινό για όλους τους εξυπηρετητές και θα μπορεί να παρακολουθεί

όλες τις TLS συνδέσεις. Η επιτυχία της επίθεσης αυτής οφείλεται στην ευπάθεια του πρωτοκόλλου όπου κατά τη χειραψία, αμέσως μετά την αποστολή του ClientMasterKey, ο εξυπηρετητής απαντά με το ServerVerify, που περιέχει το RSA κρυπτογραφημένο κείμενο, χωρίς να περιμένει πρώτα το μήνυμα επιβεβαίωσης από το πελάτη ClientFinished. Επιπλέον, η επιλογή του εξαγόμενου συμμετρικού αλγορίθμου RC2 ή RC4 με 40-bit κρυπτογράφησης, οδηγεί στην κρυπτογράφηση μόνο των 5 bytes του master_secret, αφήνοντας τα υπόλοιπα 11 σε απλό κείμενο (Aviram, N., Schinzel, S., Somorovsky, J., Heninger, N., Dankel, M., Steube, J., Valenta, L., Adrian, D., Halderman, J., A., Dukhovni, V., Käsper, E., Cohney, S., Engels, S., Paar, C., Shavitt, Y., 2016, pp. 2-8) (Dukhovni V., Käsper, E., 2016).

# Κεφάλαιο 7

# Επίλογος

Η εξασφάλιση της ακεραιότητας, εμπιστευτικότητας και διαθεσιμότητας στο διαδίκτυο είναι αδιαπραγμάτευτος στόχος και οδηγούν στην εξασφάλιση της ασφάλειας στο διαδίκτυο. Η συνεχής ενημέρωση του κοινού για τους κινδύνους που ελλοχεύουν στο διαδίκτυο και για τους τρόπους αποφυγής τους, μπορεί να μειώσει το μέγεθος της ζημίας μιας επίθεσης.

Στα πλαίσια της Μεταπτυχιακής Διατριβής υλοποιήθηκαν, με επιτυχία, τρεις αξιόλογες επιθέσεις, με τη χρήση του εργαλείου FlexTLS, χωρίς τη χρήση κάποιας ιδιαίτερης υπολογιστικής δύναμης, γεγονός που αποδεικνύει ότι οι επιθέσεις αυτές μπορούν να υλοποιηθούν με λίγες γνώσεις και σχεδόν καθόλου κόστος. Ακόμα και οι δυο μεγάλες επιθέσεις FREAK και Logjam δε χρειάστηκαν περισσότερα από 100 αμερικάνικα δολάρια στο Amazon EC2, για να μπορέσουν να διασπάσουν τα κλειδιά. Η ευπάθεια που έγινε η αιτία να υλοποιηθούν οι δυο επιθέσεις συνδέεται άμεσα με την Αμερικάνικη Κυβέρνηση, γεγονός που γεννά εύλογα ερωτήματα όσον αφορά το γεγονός της ιδιωτικότητας στο διαδίκτυο.

Στις πολλές ευπάθειες και αδυναμίες που είχε το πρωτόκολλο TLS/SSL που οδήγησαν είτε σε μικρής, είτε σε μεγάλης κλίμακας επιθέσεις, έρχεται το Μάρτιο του 2016 να προστεθεί η επίθεση του DROWN. Η επίθεση αυτή οφείλεται στην υποστήριξη από μερικούς εξυπηρετητές της απαρχαιωμένης έκδοσης SSL v2.0. Ο επιτιθέμενος μπορεί να υποβιβάσει τη σύνδεση από TLS σε SSL v2.0 και να μπορέσει να διασπάσει ευκολότερα το ιδιωτικό κλειδί του εξυπηρετητή και όταν επιτεθεί αργότερα σε στην TLS σύνδεση να μπορεί να αποκρυπτογραφήσει τα δεδομένα, αφού κατέχει το κλειδί.

Η παραπάνω επίθεση συμβαίνει εν τω μέσο του σχεδιασμού του νέου πρωτοκόλλου TLS v1.3. Το πρωτόκολλο ξεκίνησε να σχεδιάζεται με αφορμή τις επιθέσεις FREAK και Logjam, προσθέτοντας καινούργια μηνύματα ή αλλάζοντας το περιεχόμενο σε ήδη υπάρχοντα μηνύματα, στη διαδικασία της χειραψίας. Ο σχεδιασμός και η υλοποίηση του καινούργιου πρωτοκόλλου, πρέπει να δώσουν ένα αποτέλεσμα πιο ανθεκτικό σε επιθέσεις, και με λιγότερες ευπάθειες. Η χρήση του FlexTLS για τον έλεγχο του και τη διαπίστωση ευπαθειών θα παίξει καθοριστικό ρόλο στον ορθό σχεδιασμό του πρωτοκόλλου, καθώς υποστηρίζει αλλαγές στα μηνύματα και είναι συμβατό με νέα πρωτόκολλα.

Μέχρι τότε όμως πελάτες και διαχειριστές των εξυπηρετητών θα πρέπει να διατηρούν ενημερωμένους τους φυλλομετρητές τους και τους εξυπηρετητές τους, αντίστοιχα, χωρίς να σκέφτονται ότι κάτι που πλέον μπορεί να μην υποστηρίζεται στον έναν απ' τους δυο δε μπορεί να οδηγήσει σε μια σοβαρή επίθεση.

# Βιβλιογραφία


Adrian, D., Bhargavan, K., Durumeric, Z., Caudry, P., Green, M., Halderman, A., J., Heninger, N., Springall, D., Thome, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanela-Beguelin, S., Zimmerman, P.,. (2015). Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. *22nd ACM Conference on Computer and Communications Security, CSS* .

Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, A., J., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E, Zanella-Beguelin, S., Zimmermann P. (2015). *Weak Diffie-Hellman and the Logjam Attack.* https://weakdh.org/ [Προσβαση: 2/5/16].

Asokan, N., Niemi, V., Nyberg, K. (2005). *Man-in-the-Middle in Tunnelled Authentication Protocols* (Τόμ. 3364). (B. Christianson, B. Crispo, J. A. Malcolm, & M. Roe, Επιμ.) Cambridge, UK: Springer Berlin Heidelberg.

Aviram, N., Schinzel, S., Somorovsky, J., Heninger, N., Dankel, M., Steube, J., Valenta, L., Adrian, D., Halderman, J., A., Dukhovni, V., Käsper, E., Cohney, S., Engels, S., Paar, C., Shavitt, Y. (2016). *DROWN: Breaking TLS using SSLv2.* https://drownattack.com/.

Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub,P-Y., Zinzindohoue. J., K. (2015). A Messy State of the Union: Taming the Composite State Machines of TLS. *2015 IEEE Symposium on Security and Privacy* , (σσ. 535 - 552 ).

Beurdouche, B., Delignat-Lavaud, A., Kobeissi, N., Pironti, A., Bhargavan, K. (2015). *FLEXTLS A Tool for Testing TLS Implementations.* USENIX Workshop on Offensive Technologies (WOOT).

Bhargavan, K, Fournet, C., Kohlweiss, M., Pironti, A., Strub, P., Y. (2013). *Implementing TLS with verified cryptographic security.* IEEE Symposium on Security and Privacy.

Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S., Zinzindohoué, J., K., Beurdouche, B., Kobeissi, N., Pironti, A., Strub, P., Y. (2015). *SmackTLS.* https://mitls.org/pages/attacks/SMACK [Προσβαση: 1/5/2016].

Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., Strub, P., Y. (2014). *Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS.* Oakland: IEEE Symposium on Security and Privacy.

CCITT. (1991). *X.800: Security architecture for Open Systems Interconnection for CCITT applications.* International Telecommunication Union.

Dacosta, I., Ahamad, M., Traynor, P. (2012). Trust No One Else: Detecting MITM Attacks against SSL/TLS without Third-Parties. *Computer Security – ESORICS 2012* (σσ. pp 199-216). Pisa, Italy: Springer Berlin Heidelberg.

Daemen, J., Rijmen, V. (2001). *FIBS PUB 197 - Announcing theADVANCED ENCRYPTION STANDARD (AES).* Federal Information Processing Standards.

Dierks, T., Allen, C. (1999). *Request for Comments: 2246 - The TLS Protocol Version 1.0.* Network Working Group.

Dierks, T., Rescorla, E. (2006). *Request for Comments: 4346 - The Transport Layer Security (TLS) Protocol Version 1.1.* Network Working Group.

Dierks, T., Rescorla, E. (2008). *Request for Comments: 5246 - The Transport Layer Security (TLS) Protocol Version 1.2.* Network Working Group.

Dukhovni V., Käsper, E. (2016). *An OpenSSL User's Guide to DROWN.* https://www.openssl.org/blog/blog/2016/03/01/an-openssl-users-guide-to-drown/.

Eberz, S., Strohmeier, M., Wilhelm, M., Martinovic, I. (2012). *A Practical Man-In-The-Middle Attack on Signal-Based Key Generation Protocols* (Τόμ. 7459). (S. Foresti, M. Young, & F. Martinelli, Επιμ.) Pisa, Italy: Springer Berlin Heidelberg.

Eriksson, M. (2003). *An Example of a Man-in-the-middle Attack Against Server Authenticated SSL-sessions.* International Conference on Applied Cryptography and Network Security.

Evans, D., L., Bond, P., J., Bement, A., L. (2001). *FIPS PUB 140-2: Security Requirements for Cryptographic Modules.* National Institute of Standards and Technology (NIST).

Freier, A., Karlton, P., Kocher, P.,. (2011). *Request for Comments: 6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0.* Internet Engineering Task Force (IETF).

Hickman, K. E. (1995). *The SSL Protocol.* Internet Draft.

Jager., T., Kholar., F., Schage., S., Schwenk, J.,. (2013). *On the Security of TLS-DHE in the Standard Model.* ITC-2007-216676 ENCRYPT II.

Kaliski, B., Staddon, J.,. (1998). *Request for Comments: 2437 - PKCS #1: RSA Cryptography Specifications Version 2.0.* Network Working Group, RSA Laboratories.

Kent, S. (2005α). *Request for Comments: 4302 - IP Authentication Header.* Network Working Group.

Kent, S. (2005β). *Request for Comments: 4303 - IP Encapsulating Security Payload (ESP).* Network Working Group.

Kikuchi, M. (2014). *How I discovered CCS Injection Vulnerability (CVE-2014-0224).* http://ccsinjection.lepidum.co.jp/blog/2014-06-05/CCS-Injection-en/index.html [Πρόσβαση: 2/5/2016].

Langley, A., Modadugu, N., Moeller, B. (2010). *Transport layer security (TLS) false start.* IETF Internet Draft.

Meyer, C., Schwenk, J.,. (2013). *Lessons Learned From Previous SSL/TLS Attacks - A Brief Chronology Of Attacks And Weaknesses.* Cryptology ePrint Archive, Report 2013/049.

Nadhem, J., A., Kenneth, G., P. (2013). Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. *Security and Privacy (SP), 2013 IEEE Symposium on* (σσ. 526 - 540 ). Berkeley, CA : IEEE.

OpenSSL, S. (2015). *OpenSSL - Cryptography and SSL/TLS Toolkit.* https://www.openssl.org/ [Πρόσβαση 4/5/16].

Orman, H. (1998). *Request for Comments: 2412 - The OAKLEY Key Determination Protocol.* Network Working Group.

Ray, M., Dispensa, S. (2009). *Renegotiating TLS.*

Rescorla, E. (1999). *Request for Comments: 2631 - Diffie-Hellman Key Agreement Method.* Network Working Group.

Rescorla, E., Ray, M., Dispensa, S., Oskov., N. (2010). *Request for Comments: 5746 - Transport Layer Security (TLS) Renegotiation Indication Extension.* Internet Engineering Task Force (IETF).

Rex, M. (2009). *[TLS] MITM attack on delayed TLS-client auth through renegotiation.* http://www.ietf.org/mail-archive/web/tls/current/msg03928.html [Πρόσβαση: 2/5/16].

Richardson, D. (2014). *Heartbleed Bug: What is it, Who is handling our security?* http://www.inferse.com/14435/heartbleed-bug-handling-security/ [Πρόσβαση: 4/5/16].

Ristić, I. (2014). *SSL/TLS Deployment Best Practices version 1.4.* Qualys SSL Labs.

Ristić, I. (2015). *Bulletproof SSL and TLS - Understanding and deploying SSL/TLS and PKI to secure servers and web applications.* London: Feisty Duck. Fine Technology Books.

Shirey, R. (2000). *Request for Comments: 2828 - Internet Security Glossary.* Network Working Group.

Stallings, W. (2010). *Network Security Essentials: Applications and Standards, Fourth Edition.* Prentice Hall.

Stallings, W. (2014). *Cryptography and Network Security: Principles and Practice, Sixth Edition.* Pearson.

Symantec. (2016). *SSL Certificates Support.* https://knowledge.symantec.com/support/ssl-certificates-support/index.html [Πρόσβαση: 8/5/16].

Symantec, S. R. (2015). *Critical OpenSSL vulnerability could allow attackers to intercept secure communications.* http://www.symantec.com/connect/blogs/critical-openssl-vulnerability-could-allow-attackers-intercept-secure-communications [Πρόσβαση: 4/5/16].

Turner, S., Polk, T. (2011). *Request for Comments: 6176 - Prohibiting Secure Sockets Layer (SSL) Version 2.0.* Internet Engineering Task Force (IETF).

Wikipedia, T. F. (2016). *OpenSSL.* https://en.wikipedia.org/wiki/OpenSSL [Πρόσβαση: 4/5/16].

William, M., D., Raymont, G., K.,. (1999). *FIPS PUB 46-3 - DATA ENCRYPTION STANDARD (DES).* U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION.

Βογιατζάκης, Α. Γ. (2008). *Αρχιτεκτονική Ασφάλειας Δικτύων Ενσωματομένων Συστημάτων.* Πανεπιστήμιο Παρών, Πολυτεχνική Σχολή, Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών, Διαδακτορική Διατριβή.

Γεωργιάδου, Μ., Ζιαζίας, Α. (2007). *Ασφάλεια στο Διαδίκτυο.* Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, Τμήμα Πληροφορικής, Διπλωματική Εργασία.

Γκολέμη, Ε. (2010). *Κρυπτογραφία και Εξόρυξη Δεδομένων.* Πανεπιστήμιο Πατρών, Σχολή Θετικών Επιστημών, Τμήμα Μαθηματικών, Μεταπτυχιακή Εργασία.

Κατσίδου, Μ. (2007). *Ασφάλεια Ηλεκτρονικών Συναλλαγών και Internet. Η Περίπτωση των Εικονικών Εταιριών.* Πανεπιστήμιο Μακεδονίας, Οικονομικών και Κοινωνικών Επιστημών, Δια τμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών στα Πληροφοριακά Συστήματα (M.I.S), Μεταπτυχιακή Διατριβή.

Κολοκοτρώνης, Ν. (2014). *Secure Socket Layer.* Πανεπιστήμιο Πελοποννήσου, Τμήμα Επιστήμης και Τεχνολογίας Υπολογιστών, Σημειώσεις Μαθήματος.

Κολοκοτρώνης, Ν. (2014). *Βασικά Θέματα Κρυπτογραφίας.* Πανεπιστήμιο Πελοποννήσου, Τμήμα Επιστήμης και Τεχνολογίας Υπολογιστών, Σημειώσεις Μαθήματος.

Κολοκοτώνης, Ν. (2014). *Ενότητα 6η: Ασφάλεια στο Διαδίκτυο.* Πανεπιστήμιο Πελοποννήσου, Τμήμα Επιστήμης και Τεχνολογίας Υπολογιστών, Σημειώσεις Μαθήματος.

Λιμνίωτης, Κ. (2006). *Σχεδίαση Εικονικών Δικτύων.* Τεχνολογικό Εκπαιδευτικό Ιδρυμα Λαμίας, Σχολή Τεχνολογικών Εφαρμογών, Τμήμα Πληροφορικής και Τεχνολογίας Υπολογιστών, Διπλωματική Εργασία.

Μάντζιος, Ι., Μπαντίας, Σ. (2007). *Ασφάλεια στο Διαδίκτυο.* ΤΕΙ Ηπείρου, Σχολή Διοίκησης και Οικονομίας, Τμήμα Πληροφορικής και Διοίκησης, Πτυχιακή Εργασία.

Μουρσελλά, Ε. (2013). *Η Ασφάλεια των ηλεκτρονικών συναλλαγών στο διαδίκτυο.* Ανώτατο Τεχνολιγικό Εκπαιδευτικό Ιδρυμα Κρήτης, Σχολή Διοίκησης και Οικονομίας, Τμήμα Διοίκησης Επιχειρήσεων .

Μπόζης., Ε. Ζ. (2011). *Πρωτόκολλα Αυθεντικοποίησης.* : Πανεπιστήμιο Πελοποννήσου, Σχολή Θετικών Επιστημών, Τμήμα Επιστήμης και Τεχνολογίας Υπολογιστών, Μεταπτυχιακή Διπλωματικής Εργασίας.

Μωραΐτης, Δ. Σ. (2012). *Επιθέσεις ενδιάμεσου στο πρωτόκολλο TLS.* Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών, Σχολή Θετικών Επιστημών, Τμήμα Πληροφορικής και Τηλεπικοινωνιών.

Νούκας, Κ., Παππής, Χ. (2006). *Διασύνδεση και Ασφάλεια Υπολογιστών – Τεχνολογίες Ψηφιακών Ράδιο-δικτύων Υψηλής Ταχύτητας IEEE 802.11.* Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, Πολυτεχνική Σχολή, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών, Διπλωματική Εργασία.