Digital Whisper Technologic Papers

TLS - חלק א' (הצפנות א-סימטריות, RSA הבסיס המתמטי)

מאת שחר קורוט (Hutch)

הקדמה

Transport Layer Security (או בקיצור TLS), הינו פרוטוקול האבטחה הפופולרי והחשוב ביותר של רשת אינטרנט. כמעט כל אתרי האינטרנט המוגנים באמצעים קריפטוגרפיים מסתמכים על פרוטוקולים המהווים חלק מהחבילה SSL/TLS. מסחר אלקטרוני, בנקאות מקוונת, דואר אלקטרוני, VoIP, מחשוב ענן ועוד...

TLS הוא פרוטוקול ורסטילי שמטרתו אבטחת שיחת שרת/לקוח בשיטות קריפטוגרפיות חזקות והוא אמור למנוע ציתות, זיוף, או חבלה (שינוי זדוני) של המידע העובר בין השרת והלקוח. מאפשר חיבור אנונימי, אימות שרת (חד-צדדי) או אימות דו-צדדי, תוך שמירה על דיסקרטיות ושלמות המסרים. שלוש נקודות עיקריות שהפרוטוקול אמור לתת להם מענה הם:

- פרטיות - המושגת באמצעות הצפנה סימטרית.
- אימות - המושג באמצעות תעודת מפתח ציבורי.
- אמינות - המושגת באמצעות קוד אימות מסרים.

[מתוך ויקיפדיה]

ב-TLS אנו, בין היתר, משתמשים בהצפנה סימטרית הדורשת "סוד משותף" כלשהו. סוד זה יכול להקבע על ידי הסכמה מראש של שני הצדדים, אך אחד הדברים המדהימים ב- TLS הוא שאיננו קובעים את ה"סוד המשותף" מראש אלא מעבירים אותו על גבי הרשת באופן שתוקף לא יוכל להאזין לו, דבר שנשמע מעט לא הגיוני אבל בהחלט אפשרי.

למעשה, זהו בדיוק הפתרון שנותן לנו עיקרון החלפת המפתחות שבו נעסוק במאמר זה, שבו השרת והלקוח קובעים "סוד משותף" שהוא הבסיס להצפנה הסימטרית.

על מנת להבין את TLS טוב יותר ראשית עלינו להסתכל על הפרוטוקול כפתרון לבעיית "החלפת המפתחות", כדי שנוכל להבין טוב יותר החלטתי לתת דוגמא שלא כוללת בתוכה את טכנולוגיית המחשבים.

הסיפור הקלאסי מתחיל כמובן באליס ובוב שגרים באותה מדינה ונוהגים להתכתב בדואר. שניהם יודעים שפקיד הדואר, איב, נוהג לפתוח את הדואר לפני שהוא מעביר אותו ולחטט במכתבים.

יום אחד אליס מעלה רעיון מעניין, היא שולחת חבילה לבוב ומחליטה לנעול את החבילה באמצעות מפתח, את המפתח הנוסף שמה בתוך הקופסה. אליס שולחת את הקופסה לבוב. כאשר פקיד הדואר איב מנסה לגלות את תוכן החבילה הוא לא מצליח כיוון שאין לו את המפתח.

הקופסה מגיעה לבוב, בוב נועל גם הוא את הקופסה עם המנעול שלו ושולח חזרה לאליס. כמובן שאיב עדיין לא יכול לפתוח את הקופסה שנעולה כרגע בשתי מנעולים.

אליס מקבלת את הקופסה, פותחת את המנעול שלה ושולחת חזרה לבוב. איב, עדיין אינו יכול לפתוח את הקופסה בגלל המנעול של בוב שעדין קיים עליה.

בוב מקבל את הקופסה, פותח את המנעול שלו וכעת יש את המפתח למנעול של אליס שהוא המפתח המשותף לשתיהם.

Transport Layer Security - TLS

TLS הינו פרוטוקול האחראי על פרטיות המידע העובר בין שני רכיבי רשת המעוניינים לקיים שיחה מוצפנת ולמעשה מאפשר לנו להוסיף אבטחה לפרוטוקולים אפליקטביים כמו HTTP, FTP, SNMP וכו'. הפרוטוקול מורכב משתי שכבות, **TLS Handshake** ו-**TLS Record Protocol** כאשר התקשורת מבוססת TCP.

TLS Record Protocol יוצר קישור מאובטח אשר מספק את שני המאפיינים הבאים:

- **פרטיות החיבור** - המושגת באמצעות **הצפנה סימטרית** כגון RC4 ו-AES, המפתח הפרטי עבור ההצפנה הסימטרית נקבע מראש בשכבת ה-TLS Handshake.
- **אמינות החיבור** - המושגת באמצעות **Message Authentication Code, MAC** הינו שם כולל לקבוצה של פונקציות עם מפתח סודי המשמשות לאימות (Authentication) והבטחת שלמות מסרים (Message Integrity).

  מנגנון ה-MAC משתמש בדרך כלל בפונקציות Hash (למרות שזו לא הדרך היחידה) כגון MD5, SHA1 SHA256.

ה-TLS Record Protocol למעשה מבצע אנקפסולציה (כימוס) של מספר פרוטוקולים בדר"כ של שכבות הגבוהות יותר לדוגמא HTTP.

TLS Handshake - מאפשרת לשרת וללקוח להזדהות אחד עם השני, ולסכם את אלגוריתם ההצפנה והמפתחות הקריפטוגרפיים שבהם ישתמשו, כל זה נעשה למעשה לפני ששכבת האפליקציה מתחילה את העברת המידע.

בלחיצת היד משיגים שלושה יעדים:

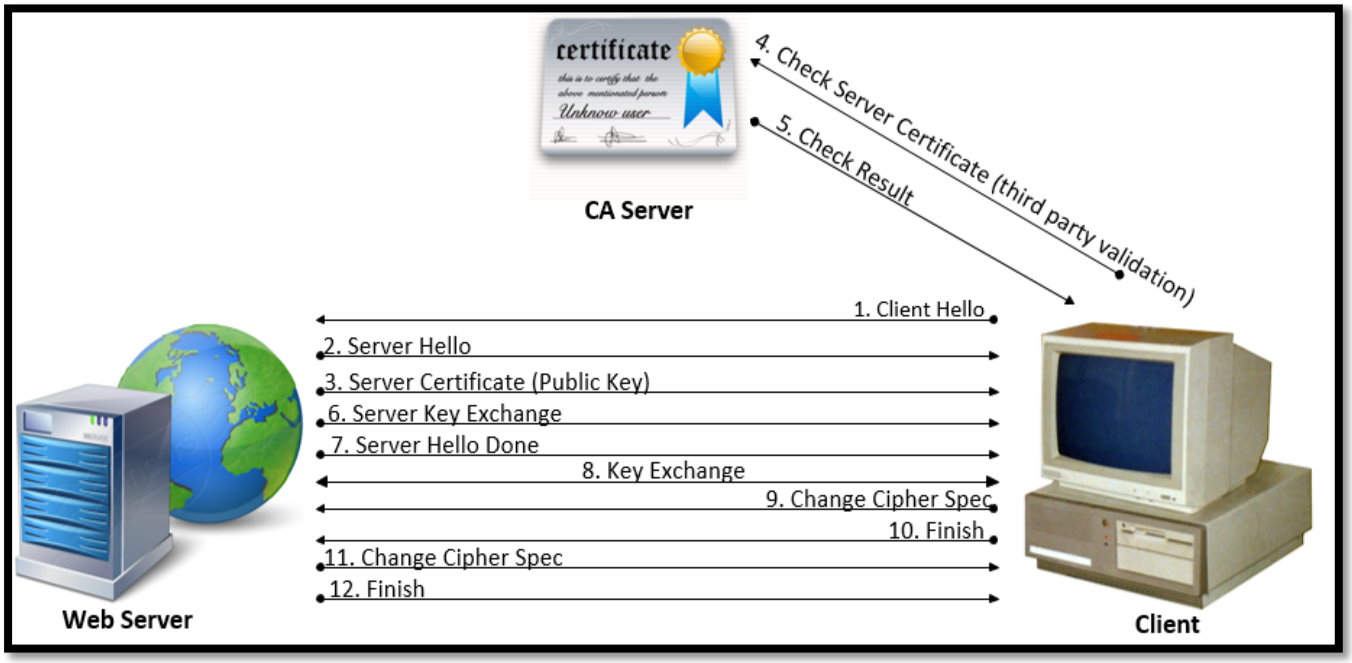
1. הצדדים מסכמים ביניהם על "חבילת צופן", ערכה של אלגוריתמים קריפטוגרפיים (וגודל מפתחות ההצפנה) שייעשה בהם שימוש לצורך הפרוטוקול.
2. הצדדים מייצרים סוד משותף על מנת לגזור ממנו מפתחות שיחה, העברת הסוד מתבצעת על ידי **הצפנה א-סימטרית**.
3. הצדדים מאמתים את זהותם. אף על פי ש-SSL תומך באנונימיות, אימות חד-צדדי או אימות דו-צדדי, מקובל בשלב זה לוודא רק את זהות השרת.

[מתוך ויקיפדיה]

TLS Handshake יוצר קישור מאובטח אשר מספק את שלושת המאפיינים הבאים:

- **הזדהות (authentication)** - של הצדדים המתקשרים, יכולה עשויה להיותלהזדהות באמצעות הצפנה סימטרית או לחלופיולהתבצע ע"י הצפנה א-סימטרית כגון RSA, DSA Diffie-Hellman וכו'
- **החלפת המפתח המשותף בוצעה באופן מאובטח** - כלומר הסוד המשותף לא ניתן לציתות אף אם תוקף ביצע בהצלחה מתקפת Man In The Middle על החיבור בין שתי הצדדים.
- **החלפת המפתח המשותף בוצעה באופן אמין** - כלומר אף תוקף לא יוכל לערוך את התקשורת בעת החלפת המפתח מבלי שהדבר יזוהה על ידי שני הצדדים המתקשרים.

נחלק את התקשורת לארבעה שלבים עיקריים (על מנת לאפשר הבנה טובה יותר):



- שלב א' - כולל את שלבים 1,2,3,6,7.
- שלב ב' - כולל את שלבים 4,5 (שלב ב' קוטע את שלב א', נתייחס לזה בהמשך)
- שלב ג' - כולל את שלב 8.

- שלב ד' - כולל את שלבים 9-12.

Basic TLS Handshake

מושגי בסיס:

- **Cipher suite -** כאשר מתבצע Handshake בתהליך ה-TLS, הלקוח משתף עם השרת את ה- cipher suite list כדי להבהיר לשרת באיזה סוגי הצפנות תומך הלקוח, השרת לאחר מכן שולח ללקוח את ה- cipher suite הנבחר.
- בפעולה זו נקבעים מספר פרמטרים:
- **Key exchange algorithm** - השרת והלקוח בוחרים את האלגוריתם שאיתו יבצעו הזדהות זה מול זה.
- **Bulk encryption algorithm** - השרת והלקוח בוחרים את השיטה שבה יוצפן המידע בשלב ההצפנה הסימטרית.
- **Message Authentication Code** - קוד אימות מסרים - Message Authentication Code בקיצור **MAC**, הוא שם כולל לקבוצה של פונקציות עם מפתח סודי המשמשות לאימות (Authentication) והבטחת שלמות מסרים (Message Integrity). פונקציית MAC מקבלת מפתח סודי ומסר באורך שרירותי ומפיקה פיסת מידע קצרה הנקראת תג אימות (Authenticator) והוא נשלח לצד המקבל יחד עם המידע המאומת או בנפרד. המקבל יכול בעזרת אלגוריתם מתאים לוודא באמצעות התג שקיבל שהמסמך אותנטי. אלגוריתם קוד אימות מסרים הוא סימטרי במובן שהשולח והמקבל חייבים לשתף ביניהם מראש מפתח סודי, באמצעותו המקבל יכול לוודא שהמסמך הגיע מהמקור שהוצהר וכי לא נעשה כל שינוי בתוכנו במהלך ההעברה. היות שלא ניתן להכין תג אימות מתאים ללא ידיעת מפתח האימות הסודי, אם נעשה שינוי כלשהו בתוכן המסר לא יצליח היריב לשנות גם את התג בצורה מתאימה ולכן המקבל יבחין בשינוי בסבירות גבוהה מאוד, יידחה את המסר המזויף על הסף ויעביר הודעה מתאימה לשולח.

[מתוך ויקיפדיה]

השרת והלקוח קובעים את סוג ה-MAC בו ישתמשו.

לדוגמא TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_RSA - היא ה-Key exchange algorithm, AES_128_GCM - היא סוג ההצפנה הסימטרית, ו-SHA256 הינו סוג ה-MAC.

- **<u>Digital certificate</u>** - או בשמו השני public key certificate, שיטת הצפנה זו למעשה עובדת בצורה כזאת שמפתח ההצפנה שונה ממפתח הפענוח (נקרא גם הצפנה א-סימטרית, למפתח הציבורי והפרטי קשר מתמטי). כלומר כל משתמש מייצר זוג מפתחות: מפתח ציבורי (Public key) שהוא מפתח הצפנה הפומבי, הנגיש לכל ומפתח פרטי (Private key) מתאים, הנשמר בסוד ומשמש לפענוח.

ההתאמה היא חד-חד ערכית (לכל מפתח ציבורי קיים אך ורק מפתח פרטי יחיד המתאים לו ולהפך). כדי להצפין מסר בשיטה זו על המצפין להשיג לידיו עותק אותנטי של המפתח הציבורי של המקבל.

- בטיחות שיטת מפתח ציבורי נשענת על הקושי שבחישוב המפתח הפרטי מתוך המפתח הציבורי. מסיבה זו מכונה שיטה זו א-סימטרית, לעומת לשיטת הצפנה סימטרית שבה מפתח הפענוח זהה למפתח ההצפנה.
- **Public key infrastructure -** או בקיצור PKI, היא למעשה אוסף של תוכנות, נהלים וכל הפרוצדורות אשר כוללות ניהול של ה-Digital certificate.

  ה-PKI מורכב ממספר גורמים Certificate Authority - CA, Registration Authority - RA, VA - validation authority.
- **CA -** משמש כ-root of trust, בתשתית ה-PKI ומספקת שירותי זיהוי למחשבים.

כעת, לאחר שהבנו את מושגי הבסיס - נמשיך להסבר שלב א'.

הסבר תהליך ה-TLS בצורה מופשטת:

ניקח לדוגמא חיבור לאתר אינטרנט על מנת לפשט את התהליך מחשבתית, הלקוח פונה לאתר באמצעות HTTPs כלומר פותח חיבור TCP מול השרת, לאחר שהושלם חיבור TCP, הלקוח שולח Client Hello (שלב א') שבו למעשה הוא מבקש מהשרת לעבור לקישור מוצפן (ולמעשה הלקוח גם מציג לשרת בבקשה זו את סוגי ההצפנות שבהם הוא תומך).

השרת בתמורה שולח Server Hello, הודעה שבה הוא למעשה מסכים לעבור לקישור מוצפן בהתאם להצפנה שהשרת בחר (כמובן ששתיהם הצדידים צריכים להכיר אותה), בנוסף הוא שולח את ה- Public Key שלו שזאת למעשה תעודה שהלקוח יאמת מול צד שלישי כדי להוכיח שאף גורם זר אינו מנסה להתחזות לשרת שאילו ניסה להתחבר (בשלב ב').

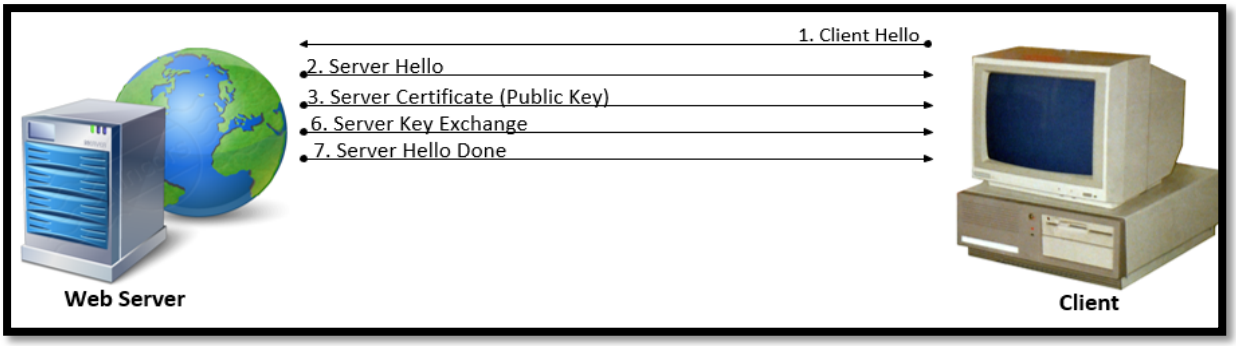
לאחר מכן הלקוח והשרת מבצעים "החלפת מפתחות" שנקרא גם שלב ההצפנה הא-סימטרית (שלב ג') שבו למעשה הם מסכימים על מפתח סודי שאיתו יצפינו את כל התעבורה, הם עושים זאת על גבי תווך שאינו מוצפן ובכל זאת לא מעבירים ממש את ה"מפתח הסודי" בתווך זה (יוסבר לעומק כאשר נגיע לדוגמא ל-RSA)

ולאחר מכן לאחר שיש ללקוח ולשרת מפתח משותף להצפנה הם יכולים להשתמש במפתח זה להצפנה סימטרית כלומר שימוש בהצפנות כגון AES או 3DES שדורשים "סוד משותף" מהלקוח והשרת (שלב ד').

וכך למעשה התבצע חיבור מוצפן שלא ניתן לפצח אותו גם אם תוקף האזין לתעבורה, וגם אם תוקף ניסה להתחזות לשרת.

שלב א'

הקישור למעשה מתחיל בתהליך שאנו מכירים שהוא למעשה קישור TCP בסיסי שאנו מכירים, הלקוח שולח SYN השרת מחזיר לו ACK ולאחר מכן לאחר שהלקוח שולח את ה-SYN, ACK שלו הוא מתחיל עם ה-Client Hello שהוא למעשה הבקשה הראשונה להתחיל לדבר ב-TLS.



בקשת Client Hello: בתמונה הבאה (הלקוחה מתוך Wireshark) ניתן לראות חבילת Hello ששלח הלקוח לשרת, בנוסף, ניתן לראות כי שכבת ה-Secure Socket Layer הינה השכבה שעוטפת את כל השכבות מתחתיה:

```
⊞ Frame 23208: 260 bytes on wire (2080 bits), 260 bytes captured (2080 bits) on interface 0      <---------- שכבה 2
⊞ Ethernet II, Src: LiteonTe_15:ca:fa (40:f0:2f:15:ca:fa), Dst: Cisco_fd:6f:c0 (e4:c7:22:fd:6f:c0)   ---------- שכבה 2
⊞ Internet Protocol Version 4, Src: 192.168.2.52 (192.168.2.52), Dst: 93.184.220.127 (93.184.220.127) <---------- שכבה 3
⊞ Transmission Control Protocol, Src Port: 2283 (2283), Dst Port: 443 (443), Seq: 1, Ack: 1, Len: 206 <---------- שכבה 4
⊟ Secure Sockets Layer <---------- שכבה 7
  ⊟ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 201
    ⊟ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 197
        Version: TLS 1.2 (0x0303)
      ⊟ Random
          GMT Unix Time: Mar 23, 2090 07:47:50.000000000 Jerusalem Standard Time
          Random Bytes: 3271176c8ec67b8fbab5cb48ba69989d6b97cf5115c2c66e...
        Session ID Length: 0
        Cipher Suites Length: 30
      ⊟ Cipher Suites (15 suites)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc14)
          Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc13)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
          Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
          Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
          Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
          Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
        Compression Methods Length: 1
      ⊞ Compression Methods (1 method)
        Extensions Length: 126
      ⊞ Extension: renegotiation_info
      ⊞ Extension: server_name
      ⊞ Extension: Unknown 23
      ⊞ Extension: SessionTicket TLS
      ⊞ Extension: signature_algorithms
      ⊞ Extension: status_request
      ⊞ Extension: next_protocol_negotiation
      ⊞ Extension: signed_certificate_timestamp
      ⊞ Extension: Application Layer Protocol Negotiation
      ⊞ Extension: Unknown 30032
      ⊞ Extension: ec_point_formats
      ⊞ Extension: elliptic_curves
```

נסביר על המרכיבים העיקריים:

- **Client version** - גרסאת ה-TLS שבה הלקוח מעוניין לעבוד, העדיפות תמיד תהיה החדשה ביותר שבה תומך הלקוח עם אפשרות כמובן בתמיכה בגרסאות אחורה כדי לאפשר תמיכה גם מול שרתים ישנים.
- **Random** - ערך רנדומלי שהמשתמש מייצר, הנקרא גם Cryptographic nonce.
- **Cipher suites** - הינה הרשימה של שיטות ההצפנה המועדפות והנתמכות ע"י הלקוח, ומאפשרות למעשה לשרת להבין באיזה שיטות הצפנה אפשרי לדבר עם הלקוח, בהמשך נעבור על חלק מן השיטות כדי להבין טוב יותר מה ההבדלים בינהם.

לדוגמא: Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA.

- **Session id** - לאחר שבוצעה כבר פעם אחת Handshake לקוח יכול לשלוח את ה-Session ID שקיבל מהשרת ובכך לקצר את תהליך ה-Handshake בכך שלמעשה בפרמטר זה מבקש מהשרת לחזור אל המפתח שהכינו בהתקשרות האחרונה ולמעשה למנוע החלפת מפתחות חוזרת מיותרת, בזכות המנגנון הזה ניתן לבצע Renegotiation.
- **Compression methods** - סוג הדחיסה בו מבקש הלקוח להשתמש, למשל Gzip.

לאחר שהלקוח שלח את בקשת ה-Client Hello הוא מחכה לתשובת ה-Server Hello מהשרת וכל תגובה אחרת מהשרת תגרום ל Fatal error (שגיאה קריטית).

בקשת Server Hello: השרת מחזיר ללקוח בקשת Hello, שנראת כך:

```
⊟ Secure Sockets Layer
  ⊟ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 80
    ⊟ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 76
        Version: TLS 1.2 (0x0303)
      ⊟ Random
          GMT Unix Time: Mar 17, 2037 01:01:37.000000000 Jerusalem Standard Time
          Random Bytes: 6e6d3ada1a28e5eaaf7ca14a9c4f9e4899d218e6deb9b217...
        Session ID Length: 0
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
        Compression Method: null (0)
        Extensions Length: 36
      ⊞ Extension: renegotiation_info
      ⊟ Extension: ec_point_formats
          Type: ec_point_formats (0x000b)
          Length: 4
          EC point formats Length: 3
        ⊟ Elliptic curves point formats (3)
            EC point format: uncompressed (0)
            EC point format: ansiX962_compressed_prime (1)
            EC point format: ansiX962_compressed_char2 (2)
      ⊞ Extension: SessionTicket TLS
      ⊞ Extension: status_request
      ⊞ Extension: Application Layer Protocol Negotiation
```

בקשה זו למעשה קובעת מספר דברים ללקוח:

- סוג פרוטוקול ההצפנה, TLS 1.2 במקרה הנ"ל.
- Cipher Suite - השרת למעשה לוקח אחת מן ה-Cipher suite שהציע לו הלקוח וקובע כי נשתמש בו לצורך ההצפנה, במקרה שלנו TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (לא להבהל, מבטיח לסביר על ההבדלים בהמשך!).
- Random - ערך רנדומלי שהשרת מג'נרט.

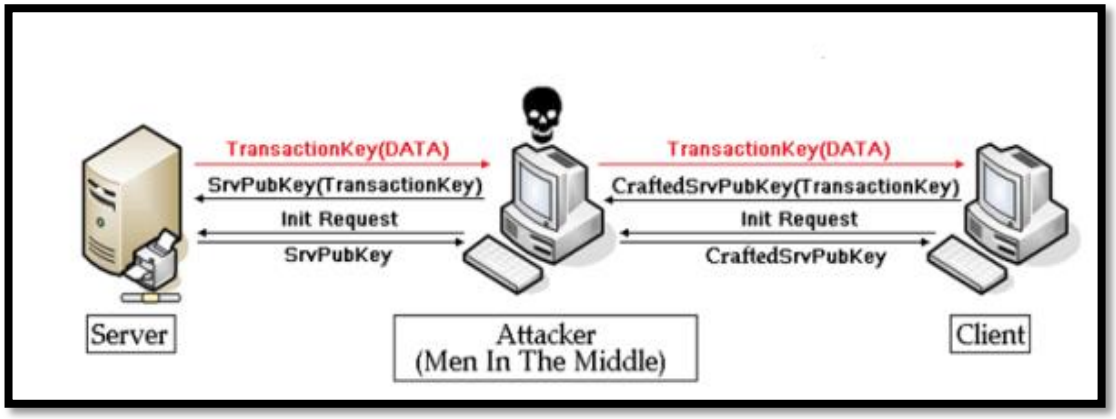
לאחר מכן השרת שולח את ה-Public Key שלו, כלומר ה-Certificate:

```
⊞ Frame 758: 1414 bytes on wire (11312 bits), 1414 bytes captured (11312 bits) on interface 0
⊞ Ethernet II, Src: Cisco_fd:6f:c0 (e4:c7:22:fd:6f:c0), Dst: LiteonTe_15:ca:fa (40:f0:2f:15:ca:fa)
⊞ Internet Protocol Version 4, Src: 93.184.220.127 (93.184.220.127), Dst: 192.168.2.45 (192.168.2.45)
⊞ Transmission Control Protocol, Src Port: 443 (443), Dst Port: 55476 (55476), Seq: 1361, Ack: 189, Len: 1360
⊞ [2 Reassembled TCP Segments (2427 bytes): #757(1275), #758(1152)]
⊟ Secure Sockets Layer
  ⊟ TLSv1.2 Record Layer: Handshake Protocol: Certificate
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 2422
    ⊟ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 2418
        Certificates Length: 2415
      ⊟ Certificates (2415 bytes)
          Certificate Length: 1282
        ⊟ Certificate (id-at-commonName=*.soundcloud.com,id-at-organizationalUnitName=Domain Control Validated)
          ⊟ signedCertificate
              version: v3 (2)
              serialNumber : 0x11216db183de5f773e78f022048c0d6a8d47
            ⊟ signature (sha256WithRSAEncryption)
                Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
            ⊞ issuer: rdnSequence (0)
            ⊞ validity
            ⊞ subject: rdnSequence (0)
            ⊞ subjectPublicKeyInfo
            ⊞ extensions: 9 items
          ⊟ algorithmIdentifier (sha256WithRSAEncryption)
              Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
            Padding: 0
            encrypted: 8272e512d4c0ea7550acc4615127859abc761aa4f29f5d8f...
```

דבר זה למעשה מבקש מהלקוח לעבור לשלב ב' שהוא בדיקת מהימנות ה-Certificate, כלומר בדיקה מול גורם שלישי שתוכיח שזהו באמת ה-Certificate ל-SoundCloud.com (האתר אילו אנו גולשים).

שלב ב'

לפני שנתחיל את שלב ב' חשוב להבין מה היה קורה ללא השימוש בו, נניח כרגע תוקפים את הלקוח שלנו במתקפת Man in the Middle:



[נלקח מתוך מאמר של אפיק קסטיאל ב-Digital Whisper SSL & Transport Layer Security Protocol]

התוקף למעשה מנצל את זה שה-Client אינו יודע לאמת את זהות השרת ומתחזה ל-Server שאילו הלקוח

מנסה להתחבר, התוקף למעשה שולח ללקוח Certificate שלו ופותח מולו קישור SSL\TLS מכיוון שעכשיו התוקף יכול לפענח את התעבורה שלנו אין לו בעיה פשוט לשמש כ-Proxy מול השרת האמיתי ולדמות למשתמש גלישה רגילה לאתר, היום בזכות הגנה זו אנו למעשה מקבלים את הודעתה השגיאה הבאה:

חיבור זה אינו בטוח

ביקשת מ־Firefox להתחבר בצורה מאובטחת אל **www.facebook.com**, אבל אנחנו לא יכולים לאמת שהחיבור שלך אכן מאובטח.

בדרך־כלל, כאשר אתה מנסה להתחבר בצורה מאובטחת, אתרים מציגים הזדהות מוסמכת כדי להוכיח שאתה אכן מגיע למקום הנכון. למרות זאת, ההזדהות של אתר זה לא ניתנת לווידוא.

מה ברצונך לעשות?

אם אתה בדרך־כלל מתחבר לאתר זה ללא בעיות, ייתכן שמישהו מנסה להזדהות בשם האתר, ורצוי שלא תמשיך הלאה לאתר.

אתר זה משתמש ב־HSTS (HTTP Strict Transport Security) כדי לציין ש־Firefox יתחבר אליו רק בצורה מאובטחת. כתוצאה מכך, לא ניתן להוסיף חריגה לאישור אבטחה זה.

קח אותי מפה!

פרטים טכניים

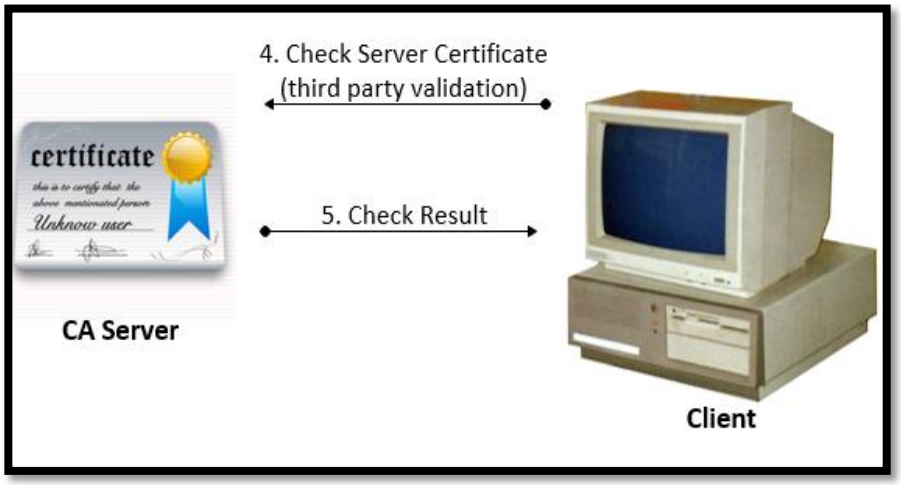
www.facebook.com עושה שימוש באישור אבטחה שאינו תקף.

האישור איננו מהיימן מאחר הוא חתום עצמית.
האישור תקף רק עבור PortSwigger.

(קוד שגיאה: sec_error_unknown_issuer)

[מתוך FilreFox]

בשלב זה למעשה הלקוח לוקח את ה-Certificate שקיבל מהשרת והוא בודק אותו מול ה- Certificate Authority, שהוא למעשה ה-Trusted third party כלומר ישות שלישית שהלקוח סומך עליה.



ישנן מספר שיטות לקבוע את מיהמנות החתימה:

- **התקנת Root Certificate** של הגורם המאשר בדפדפן המשתמש, שבאמצעותה ניתן לוודא שהחתימה של ישות כלשהי אכן נחתמה על ידי מי שמתיימר להחזיק במפתח החתימה דוגמאת מיקרוסופט.
- **שימוש ברשת אמון(Trust Web)** אפשר להקים באופן עצמאי, על ידי חתימה הדדית של משתתפי הרשת על המפתחות הציבוריים של עמיתיהם, שיכולה להתבצע דרך מה שמכונה מסיבת חתימות(Key signing party). או לחלופין בהתבסס על רשות מסחרית המספקת שרותי אימות בתשלום דוגמת VeriSign.

לאחר שקיבל את התשובה, אשר מגיעה כ-True/False הלקוח ידע אם לסיים את שלב א' ולהתקדם לשלב ג'.

שלב א' - סיום

לאחר שה-Certificate נבדק ע"י הלקוח, הוא ממשיך לקבל את הודעות השרת ה-Server Key Exchange הבקשה נראת כך:

```
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 333
    Handshake Protocol: Server Key Exchange
      Handshake Type: Server Key Exchange (12)
      Length: 329
      EC Diffie-Hellman Server Params
        Curve Type: named_curve (0x03)
        Named Curve: secp256r1 (0x0017)
        Pubkey Length: 65
        Pubkey: 0497f5bb1b05337d73b2d3098a8d7c1f8f9de944927df035...
        Signature Hash Algorithm: 0x0601
          Signature Hash Algorithm Hash: SHA512 (6)
          Signature Hash Algorithm Signature: RSA (1)
        Signature Length: 256
        Signature: 258c2f42fdd303fc0b8b5f8f8db2d90c2abe664c66663177...
```

הבקשה מכילה את ה-Public key של השרת.

מיד לאחר מכן, השרת שולח את ה-Server Hello Done שהיא למעשה הודעה פשוטה שמודיעה על המעבר לשלב החלפת המפתחות, הבקשה נראת כך:

```
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 4
    Handshake Protocol: Server Hello Done
      Handshake Type: Server Hello Done (14)
      Length: 0
```

שלב ג'

שלב זה הינו שלב החלפת המפתחות, בחלק נסביר לעומק את תהליך החלפת המפתחות כולל דוגמאות מספריות, והסברים למהם למעשה סוגי ה-Cipher Suite.

השלב נפתח כאשר הלקוח שולח אל השרת את בקשת ה-Client Key Exchange, שנראת כך:

```
⊟ Secure Sockets Layer
  ⊟ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 70
    ⊟ Handshake Protocol: Client Key Exchange
        Handshake Type: Client Key Exchange (16)
        Length: 66
      ⊟ EC Diffie-Hellman Client Params
          Pubkey Length: 65
          Pubkey: 04a59b55218a9cd074e896b60d794b8d9b6e6de8e32a572a...
```

בזאת הסתיים תהליך החלפת המפתחות ברשת, ומרגע זה שתי הצדדים כבר מחזקים במפתח סודי לפתיחת הבקשות אחד של השני, אבל כיצד הם עושים זאת?

כאשר קבענו על Cipher Suite וסיכמנו לדומא על: **TLS_RSA_WITH_3DES_EDE_CBC_SHA**

למעשה סיכמנו לעבוד ב-RSA כשיטות החלפת המפתחות (הצפנה א-סימטרית) ולאחר ה-WITH תבוא שיטת ההצפנה הסימטרית כגון 3DES, CBC היא שיטת ההצפנה (Cipher Blocks) ו-SHA היא ההצפנה בה נשתמש ב-MAC .

שיטות הצפנה א-סימטרית:

הצפנת מפתח ציבורי או הצפנה אסימטרית (Asymmetric encryption), הינה שיטת הצפנה שבה מפתח ההצפנה שונה ממפתח הפענוח. כלומר, כל משתמש מכין לעצמו זוג מפתחות: מפתח ציבורי (Public key) שהוא מפתח הצפנה הנגיש לכל ומפתח פרטי (Private key) מתאים, הנשמר בסוד ומשמש לפענוח. ההתאמה היא חד-חד-ערכית (לכל מפתח ציבורי קיים אך ורק מפתח פרטי יחיד המתאים לו, ולהפך).

[מתוך ויקיפדיה]

<u>בסיס מתמטי קצר:</u>

חזקות בסיס, נזכיר רק שזוהי חזקה (מספר כפול עצמו):

$$x * x = x^2$$

$$x * x * x = x^3$$

כמו כן נזכור שמכפלת מספר בחזקת חזקה כלשהי כפול אותו מספר בחזקה שונה, תהיה אותו המספר בחזקת סכום החזקות:

$x^a * x^b = x^{(a+b)}$

דוגמא מספרית להמחשה:

$2^3 * 2^6 = (2 * 2 * 2) * (2 * 2 * 2 * 2 * 2 * 2) = 2^9$

חשבון מודולרי:

את החשבון המודולרי אנו מכירים למשל כשאנו בודקים מה השעה בשעון כאשר השעה 17 אנו יודעים לחשב שלפי חשבון מודולרי של 12, השעה היא למעשה 5.

אנו מבצעים חילוק בשארית:

$\frac{17}{12} = 1\ (5)$

נוכל להציג זאת גם בצורה הבאה כמודולו:

$17 = 5\ (mod\ 12)$

דוגמא לפעולות בסיסיות במודולו:

אם:

$9835 = 7\ (mod\ 12)$

$$1176 = 0\ (mod\ 12)$$

אז:

$9835 + 1176 = 7 + 0\ (mod\ 12)$

כמו כן, הטריק הזה יעבוד לנו גם בכפל:

$9835 * 1176 = 11565960 = 0\ (mod\ 12)$

$9835 * 1176 = 7 * 0\ (mod\ 12)$

גורמים ראשוניים:

המשפט היסודי של האריתמטיקה קובע כי לכל מספר שלם (מלבד 0) קיימת הצגה יחידה כמכפלה של מספרים ראשוניים. תכונה זו מאפשרת לנו להתייחס למספרים הראשוניים כמעין "אטומים" של המספרים השלמים.

מספר ראשוני מוגדר כמספר שמתחלק רק בעצמו וב-1 (ללא שארית).

פירוק לגורמים ראשוניים היא שיטה לפירוק של מספרים פריקים למספרים הראשוניים אשר מרכיבים אותם. מכפלת המספרים הראשוניים הללו תביא למספר שאותו פירקנו. הנה דוגמה לפירוק המספר 12 לגורמים ראשוניים. אנו יודעים ש-3 ו-4 הינם כופלים (גורמים) של 12:

$12 = 4 * 3$

3 הוא ראשוני, אז לא נפרק אותו; אך 4 אינו ראשוני, והוא מתחלק פעמיים ל-2.

$4 = 2 * 2$

ומכאן, ניתן לראות כי הגורמים הראשוניים המרכיבים את 12 הינם: 2,2,3.

[מתוך ויקיספר]

אם נרצה לפרק את המספר 1176 לגורמיו הראשוניים נוכל להגיע לכך באופן הבא:

$1176 = 2^3 * 3^1 * 7^2$

או:

$7 * 7 * 3 * 2 * 2 * 2 = 1176$

מחלק משותף מקסימלי (gcd קיצור של greatest common divisor) של שני מספרים שלמים הוא המספר הגדול ביותר שמחלק את שניהם. למשל:

$\gcd(15,10) = 5$

$\gcd(18,10) = 2$

אם שני המספרים גדולים מידי כדי שנוכל לחשב להם גורם משותף מקסימלי נשתמש בשיטה הבאה:

נחלק את המספרים לגורמיהם הראשונים:

$\gcd(1176, 6860) = \gcd((2^3 * 3^1 * 7^2), (2^2 * 5^1 * 7^3)) = 2^2 * 7^2$

ועל כן:

$\gcd(1176,6860) = 196$

אני ממליץ לקרוא את ההרחבה בסוף המאמר על אלגוריתם אוקלידס, החלטתי לא לשים אותו כאן משום שהוא אינו קריטי להבנה באופן כללי, אלגוריתם אוקלידס הוא אלגוריתם המאפשר בהינתן שני מספרים טבעיים למצוא את המחלק המשותף המקסימלי.

פונקציית אויילר (Euler's totient function), פונקציה זו נסמן בעזרת האות היוונית פי (φ או באות הגדולה ϕ) ונסמן אותה בצורה הבאה ϕ(n) כאשר N הוא מספר טבעי. הפונקציה למעשה מחשבת את כמות המספרים אשר זרים (אין להם מכנה משותף כלשהו) למספר כלשהו לדוגמא:

$$\varphi(6) = 2$$

משום שהמספרים היחידים שזרים ל-6 הם 1 ו-5.

דוגמא נוספת:

$\varphi(9) = 6\ \{1,2,4,5,7\}$

המספר 6 אינו בקבוצה משום שהוא מכיל את 2 ו-3 כאשר 3 אינו זר ל-9.

הנה ההגדרה בקטע קוד ב-C על מנת לעשות את המשפט פשוט יותר:

```
phi = 1;
for (i = 2 ; i < N ; ++i)
    if (gcd(i, N) == 1)
        ++phi;
```

מקרה פרטי של משפט אויילר אשר מתקיים רק במספרים ראשוניים נקרא "**המשפט הקטן של פרמה**", על פי משפט זה, כל מספר ראשוני אשר נפעיל עליו את פונקציית אויילר, יחזיר לנו את המספר הראשוני פחות 1 (מפני שמספר ראשוני מתחלק רק בעצמו וב-1)

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

לדוגמא, 17 הינו מספר ראשוני ועל כן:

$$\varphi(17) = 17 - 1\ \{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16\} = 16$$

משפט אויילר:

שני מספרים נקראים זרים או ראשוניים ביחס אחד לשני, אם המחלק המשותף המרבי שלהם הוא 1.

פונקציית אוילר המסומנת ϕ(n) מייצגת את מספר השלמים בטווח [1, n-1] הזרים ל-n (או הראשוניים ביחס ל-n). כלומר שהמחלק המשותף המרבי שלהם עם n הוא 1.

אם n ראשוני אזי כל המספרים עד n-1 זרים ל-n, כיוון שמספר ראשוני אינו מתחלק באף אחד מהמספרים הנמוכים ממנו מעצם ההגדרתו.

משפט אוילר קובע שעבור כל שלם a שזר ל-n מתקיים:

$$a^{\varphi(n)} = 1\ (mod\ p)$$

כאשר p מספר ראשוני מתקבל כמקרה פרטי המשפט הקטן של פרמה (הצבנו בתוך פונקציית אויילר):

$$a^{(p-1)} = 1\ (mod\ p)$$

לכל a שלא מתחלק ב-p.

הופכי כפלי מודולרי:

נגדיר מספר כהופכי כאשר מכפלתו במספר אחר תתן את התוצאה 1, נוכל לממש זאת בעולם המספרים הרציונלים (כל מספר שניתן ליצג אותו ע"י שבר למשל 2 יכול להיות 4:2) באופן הבא:

נניח ו-a ו-b הינם שתי מספרים רציונלים המשוואה $a * b = 1$ תתקיים כאשר $a = \frac{b}{1}$, אומנם כאשר נעסוק בחשבון מודולרי שבו החשבון מתבצע עם מספרים שלמים (ללא שברים) בלבד, עדיין קיים מושג הכפל ההופכי אך הוא שונה מעט ההגדרה הינה עדיין $a * b = 1$ רק כאשר הפעם הכפל הוא מודולרי $a * b = 1\ (mod\ n)$.

לדוגמה, מודולו 9, ההופכי הכפלי של 2 הוא 5, מכיוון ש:

$$2 * 5 = 1 \ (mod\ 9)$$

נוכל להציג זו בצורה הבאה:

$$5 = \frac{1}{2} \ (mod\ 9)$$

ונכפול את שתי האיברים ב-3:

$$5 * 3 = \frac{3}{2} \ (mod\ 9)$$

אפשר לבדוק שאכן 5 הוא ההופכי של 2 בדוגמה זו, כלומר אחרי שנבצע פעולה והיפוכה נקבל בחזרה 3.

$$\frac{5}{2} * 3 = 3 \ (mod\ 9)$$
$$5 * 3 = 3 \ (mod\ 9)$$

כעת, עם הרקע התאורטי שביססנו נוכל להסביר את RSA.

RSA (Rivest-Shamir-Adleman)

האלגוריתם אשר אחראי על ג'נרוט המפתח הוא למעשה החלק המוסבך ביותר ב- RSA, מטרתה היא למעשה לג'נרט את המפתח הציבורי והמפתח הפרטי, ואלו שלביו:

1. הגרלה של מספרים ראשוניים - מוגרלים שתי מספרים ראשוניים p ו-q, המספרים הללו צריכים להיות גדולים מאוד (לפחות 1024 ספרות!)
2. Modulus - מודולו n, שלמעשה ערכו יהיה p*q .
3. פונקציית אויילר - פונקציית אויילר φ (n) מחושבת.
4. Public Key - מספר ראשוני נמצא בין המספר 3 ל-φ(n) ואשר ה-gbc שלו ושל φ (n) יהיה 1 (יוסבר במהשך)
5. Private Key - הינו מפתח יחיד אשר מתאים ל-Public Key ורק בעזרתו ניתן לפענח את הטקסט המוצפן.

הגרלה של מספרים ראשוניים:

הכרחי ש-RSA יעבוד עם מספרים גדולים ראשוניים על מנת לייצר מפתח חזק שיהיה קשה לשחזר את יצירתו, ככל שהמספרים יגדלו כך פענוח ה-RSA יהיה קשה יותר. כדי למצוא מספרים ראשוניים בקלות יחסית נוכל להשתמש באלגוריתם מילר-רבין ועל ידיו להגריל מספרים ראשוניים p ו-q.

מודולו:

כאשר בידינו שני המספרים הראשוניים שהשגנו, חישוב המודולו הוא די פשוט כפי שאמרנו:

$n = p * q$

פונקציית אויילר:

RSA Function Evaluation - פונקציה אשר לוקחת את N ומחשבת את פונקציית אויילר שלו, מכיוון שאנו יודעים שלי ה"משפט הקטן של פרמה" נוכל לחשב את פונקציית אויילר עבור כל אחד מן הראשונים כך:

$\varphi(n) = q - 1$

$\varphi(n) = p - 1$

ועל כן פונקציית אויילר של שניהם תהיה:

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

Public Key:

את המפתח הציבורי נסמן ב-e. הוא מספר ראשוני בין 3 לתוצאת פונקציית אווילר שחישבנו, כמו כן משום ש-3 יכול להפוך את כל המפתח שלנו לחלש, בדר"כ נתחיל ב-65537. לאחר שבחרנו את המפתח הציבורי eנבדוק האם ה-gcd שלו עם n של פונקציית אויילר הוא 1 (בדיקה זו למעשה מספקת לנו מידע האם המספרים הם Coprime, כלומר ראשוניים ביחס אחד לשני), אם כן נמשיך הלאה, ובמידה ולא יחושב e חדש.

את המפתח הציבורי נרשום בצורה הבאה כאשר n הוא תוצאת (כח החישוב) פונקציית אויילר, ו-e הוא המפתח הציבורי:

(e,n)

כאשר המפתח הציבורי ישמש אותנו להצפנה:

Encryption: $m^{e(mod\ n)} = c$

Private Key:

את המפתח הפרטי נסמן ב-d, משום שאנו עובדים עם מספרים ראשוניים נוכל לדעת כי אם d שונה מ-e וראשוני לו, נוכל להשתמש במשפט אויילר ולקבוע:

e * d = 1 mod φ(n)

כאשר המפתח הפרטי ישמש אותנו לפענוח:

Decryption: $c^{d(mod\ n)} = m$

שימו לב כי גם המפתח הפרטי נשמר כ-(d,n), ואלו הם למעשה זוג המפתחות של השרת.

השרת מצפין את ההודעה (M) ושולח את C (הטקסט המוצפן ללקוח). כעת, הלקוח לוקח את ה- Public Key ומצפין את ההודעה שלו. ההודעה של הלקוח היא למעשה ה-Pre-MasterSecret הודעה זו מסומנת כ-m ומוצפנת לפי המשוואה שמוצגת מעלה. לאחר מכן הוא שולח את C (הטקסט המוצפן) לשרת .

ה-Pre-MasterSecret צריך להיות באורך מינימלי של 48 והוא למעשה נוצר במחולל מספרים פסאודו-אקראיים קריפטוגרפי (Cryptographically secure pseudorandom number generator) או בקיצור:

CSPRNG. רכיב זה בדרך כלל נמצא במערכת ההפעלה והוא אמור לספק לנו ערכים רנדומילים לצורכי הצפנה.

ה-Pre-MasterSecret ישמש לנו בסיס להצפנה הסימטרית.

דוגמא:

הכרחי ש-RSA יעבוד עם מספרים ראשוניים גדולים. זאת על מנת לייצר מפתח שיהיה מספיק קשה לשחזר את יצירתו ע"י מתקפות כגון Brute Force, ככל שהמספרים יגדלו כך פענוח ה-RSA יהיה קשה יותר. על מנת למצוא מספרים ראשוניים גדולים בקלות יחסית, נוכל להשתמש באלגוריתם מילר-רבין ועל ידיו להגריל מספרים ראשוניים p ו-q.

לצורך הדגמה נבחר את המספרים הראשוניים הענקיים הבאים:

p=11

q=13

המודולו למעשה יהיה כפולה של שני המספרים הראשוניים ואותו נסמן ב-n:

n = q*p, n = 13*11, n=143

נחשב את פונקציית אויילר, בעזרת המשפט הקטן של פרמה, כפי שהוסבר קודם:

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

$\varphi(n) = (13-1) * (11-1),$

$\varphi(n) = 120$

לאחר מכן נצטרך לבחור מספר אשר המחנה המשותף הגדול ביותר שלו עם 120 הוא 1, נבחר למשל במספר הראשוני 7:

e = 7

את המפתח הציבורי נשלח ללקוח בצורה הבאה:

(e,n)

כאשר המפתח הציבורי ישמש את הלקוח להצפנה כך :

Encryption: $m^{e(mod\ n)} = c$

את המפתח הפרטי נסמן ב-d. משום שאנו עובדים עם מספרים ראשוניים נוכל לדעת כי אם d זר ל-e וראשוני לו, נוכל להשתמש במשפט אויילר ולקבוע:

$m^{e*d} = m\ (mod\ n)$

כאשר המפתח הפרטי ישמש אותנו לפענוח:

Decryption: $c^{d(mod\ n)} = m$

כפי שהזכרנו קודם משפט אוילר קובע שעבור כל שלם m שזר ל n-מתקיים:

$$m^{\varphi(n)} = 1\ (mod\ n)$$

נשתמש כעת במספר טריקים מתמטיים על מנת להקל על החישוב שלנו, אנו יודעים שכאשר נקח את המספר 1, לא משנה באיזה חזקה נעלה אותו תמיד הוא ישאר 1.

בעולם המודולרים הדבר ישאר זהה, ועל כן נוכל לכפול את φ(n) ב-k ועדיין נקבל 1, ועל כן:

$$m^{(k*\varphi(n))} = 1\ (mod\ n)$$

כמו כן אנו יודעים שכאשר מתקיימת משוואה נוכל לכפול את שתי הצדדים של המשווה באותו גורם, והוא עדיין לא ישנה את התוצאה. כלומר, מותר לנו להכפיל את שתי הצדדים ב-n.

$$m * m^{(k*\varphi(n))} = m\ (mod\ n)$$

או:

$$m^{(k*\varphi(n)+1)} = m\ (mod\ n)$$

למעשה, אנו רוצים ליצור קשר בין e ל d-כך ש:

$$m^e * d = m\ (mod\ n)$$

כאשר d ישמש לנו כמפתח הפענוח נוכל להשוות בין שתי המשוואות ונקבל ש:

$$m^e * d = m^{(k*\varphi(n)+1)}$$

נפעיל על כל המשוואה שורש של m ונקבל את המשוואה:

$$d * e = k * \varphi(n) + 1$$

או:

$$d = \frac{(k * \varphi(n) + 1)}{e}$$

שימו לב שהנתון שחסר לתוקף הינו φ (n), שכן קל מאוד לחישוב אם אתה יודע את המספרים הראשוניים אך קשה לבצע פירוק לגורמים של n כדי לדעת את φ(n).

על מנת לפענח את ההודעה, עלינו לחשב את d, ע"י:

$$d = \frac{(k * (p-1)(q-1) + 1)}{e}$$

בהנחה ש-K=6 (השרת יצטרך לחשב את ערכו של K כאשר K נותן תוצאה שלמה בלבד, אך חישוב זה קל יחסית משום השימוש בפעולת כפל בלבד):

p=11, q=13, n=143

φ(n) = 120

e = 7

$$d = \frac{(6 * 120 + 1)}{7} = 103\,, d = 103$$

כעת, שכל הנתונים בידינו נגיד והלקוח רצה לשלוח את ההודעה "A" לשרת, נתרגם אותה ל-41 (Hex) למשל, הלקוח מצפין בצורה הבאה:

$$m^e \ (mod\ n) = c$$

ובדוגמא שלנו:

$$41^7 \ (mod\ 143) = 194754273881 \ (mod\ 7) = 24, \qquad c = 24$$

והשרת יפענח ע"י החישוב: $c^d \ (mod\ m) = n$ באופן הבא:

$$103^{24} \ (mod\ 143) = 1.451302e + 142 \ (mod\ 143) = 41$$

(Diffie-Hellman) DHE

פרוטוקול דיפי-הלמן (Diffie-Hellman) הינו הפתרון המעשי הראשון לבעיית הפצת המפתחות, ביתר פירוט לבעיית "שיתוף מפתח". הפרוטוקול מאפשר לשני משתתפים שלא נפגשו מעולם ואינם חולקים ביניהם סוד משותף כלשהו מראש, להעביר ביניהם מעל גבי ערוץ פתוח (שאינו מאובטח) סוד כלשהו כך שאיש מלבדם אינו יודע. פרוטוקול דיפי-הלמן מתמודד עם בעיה זו בשיטה אסימטרית. הפרוטוקול פוטר אותם מהצורך לשמור מפתחות הצפנה סודיים לאורך זמן; תחת זאת המצפין יכול להכין מפתח הצפנה ארעי, להעבירו באמצעות הפרוטוקול לצד השני ואז התקשורת ביניהם יכולה להיות מוצפנת באמצעות צופן סימטרי מהיר כמו AES כאשר מפתח ההצפנה הוא הסוד המשותף או מפתח אחר שנגזר ממנו באמצעות פונקציה מוסכמת ובגמר השימוש בו המפתח מושמד.

ביטחון הפרוטוקול מסתמך על הקושי שבפתרון בעיית דיפי-הלמן (להלן) הדומה לבעיית הלוגריתם הדיסקרטי. הגרסה המתוארת מספקת הגנה על סודיות המפתח המשותף כנגד יריבים פסיביים המסוגלים לצותת לערוץ התקשורת בלבד. היא אינה מספקת הגנה מפני יריב אקטיבי המסוגל ליירט, לחסום או ל'הזריק' מסרים כרצונו. למעשה, הפרוטוקול אינו מספק מה שקרוי "אימות זהויות המשתתפים"

[מתוך ויקיפדיה]

למעשה יש לנו את הבסיס המתמתי המספק מהצפנת RSA ועל כן נוכל להביר אותו בהתבסס על המושגים שלמדנו קודם כדי להבין את הרעיון הבסיסי. למעשה גם הצפנה זאת עובדת בעולם המספרים המודולארים, נסביר תחילה עם מספרים קטנים כדי להבין את החלפת המפתחות.

ראשית אליס ובוב צרייכים לבחור שתי מספרים ראשוניים אשר ישמשו אותם בתהליך ההצפנה, למשל 3 ו-17.

אליס ובוב ישתמשו במשוואה הבאה:

$$3 \ mod\ 17$$

לאחר מכן כל אחד מהם בוחר מספר ראשוני, למשל:

אליס: 15 ובוב: 13.

אליס לוקחת את המספר שלה ומציבה אותו כחזקה של 3 כך:

$3^{15}\ (mod\ 17) = 6$

בוב עושה את אותו התהליך:

$3^{13}\ (mod\ 17) = 12$

כעת, אליס לוקחת את תוצאת המשוואה שלה (6) ושולחת אותה לבוב. בוב גם כן שולח את התוצאה שלו (12) לאליס.

בינתיים, נראה מה הנתונים שגורם צד שלישי (איב) יכל לאסוף בעת העברות המידע בין אליס לבוב. כרגע המידע בידו של איב הוא:

3 mod 17

Alice solution: 6

Bob solution: 12

לאחר שאליס ובוב העבירו בינהם את התוצאות הם מבצעים את המהלך הבא:

אליס לוקחת את הפיתרון של בוב 12 ומעלה אותו בחזקת המספר הראשוני המקורי שבחרה, 15 כך:

$12^{15}\ (mod\ 17) = 10$

בוב לוקח את הפיתרון של אליס 6 ומעלה אותו בחזקת המספר הראשוני המקורי שבחר, 13 כך:

$6^{13}\ (mod\ 17) = 10$

שני הצדדים קיבלו את המפתח המשותף ללא העברתו ברשת, ומבלי שאיב יהיה מסוגל לחשבו גם משום שהנתונים מספרים הראשוניים שבחרו (13 ו-15) לא נשלחו מעולם, אך מדוע אליס ובוב מקבלים את אותה התשובה?

למעשה אליס ובוב עשו חישוב זהה לחלוטין רק בסדר טיפה שונה, נסביר:

כאשר אליס מחשבת את:

$12^{15}\ (mod\ 17)$

היא למעשה מחשבת את ה-12 שבוב חישב קודם בצורה הבאה:

$3^{13}\ (mod\ 17) = 12$

ולאחר מכן מעלה אותו בחזקת 15 בצורה הבאה, כך שנוכל להציג זאת בצורה הבאה:

$(3^{13})^{15}\ (mod\ 17)$

ובוב למעשה מחשב 6 שהתקבל מאליס שחישבה בצורה הבאה:

$3^{15}\ (mod\ 17)$

ולאחר מכן מעלה אותו בחזקת 15 בצורה הבאה, כך שנוכל להציג זאת בצורה הבאה:

$$(3^{15})^{13}\ (mod\ 17)$$

כפי שכבר הוזכר קודם במאמר אין חשיבות לסדר שבו מבוצעות החזקות, ועל כן התוצאה תהיה זהה.

בעיית הלוגריתם הדיסקרטי

בעיית הלוגריתם הדיסקרטי היא בעיה באלגברה חישובית, שמהותה מציאת המעריך (חזקה) x של הערך a^x , כאשר a הינו **איבר בחבורה**; זאת בדומה לפונקציית הלוגריתם הרגילה, המחשבת את x כאשר a הינו מספר ממשי.

בעיה זו דומה לבעיית פירוק לגורמים של מספר שלם, בכך ששתיהן קשות (לא ידוע על אלגוריתם יעיל לפתרונן), ושתיהן משמשות מרכיב מהותי בפרוטוקולים של הצפנה, ובפרט בפרוטוקול דיפי-הלמן.

כלומר נגיד אציג בפניכם את המשוואה:

$$3^a\ (mod\ 17) = 2$$

לא מתאפשרת דרך מהירה לגלות מהו ערכו של a אלא בעזרת brute force, כלומר נסיון של כל המספרים עד להצלחה וזוהי גם הסיבה שאיב לא יהיה מסוגל לפצח את ההצפנה בזמן סביר כאשר מדובר בשני מספרים ראשוניים גדולים.

שלב ד'

חלק זה יפורט בהרחבה בחלק ב' של המאמר, אך להשלמת ההבנה אסביר את המשך התהליך ההתקשרות של TLS.

הצפנה סימטרית:

בקריפטוגרפיה, הצפנה סימטרית (Symmetric Encryption) או צופן סימטרי הוא אלגוריתם הצפנה שבו משתמשים במפתח הצפנה יחיד הן להצפנה של הטקסט הקריא והן לפענוח של הטקסט המוצפן. בפועל המפתח הוא בדרך כלל סוד משותף לשנים או יותר משתתפים ובדרך כלל מתאים לכמות מוגבלת של נתונים. הסיבה שהצופן נקרא סימטרי היא כי נדרש ידע שווה של חומר סודי (מפתח) משני הצדדים.

[מתוך ויקיפדיה]

באופן כללי, כאשר סיימנו את חלק ג' עברנו את שלב ההצפנה הא-סימטרית שלמעשה מאפשרת ללקוח לתקשר עם השרת ולהעביר "סוד משותף" על גבי הרשת ללא אפשרות של תוקף צד שלישי לפנענח את המידע בזמן סביר (כפי שראינו ב-RSA במספרים קטנים ניתן לתקוף את הפרוטוקול ולנסות לפצח אותו אך במספרים ראשוניים גדולים מאוד פעולה זו תקח הרבה מאוד זמן).

הלקוח יצטרך לקבוע את ה"סוד המשותף", או כפי שהזכנו אותו קודם ה-Pre-MasterSecret אשר שישמש את השרת והלקוח כבסיס להצפנה הסימטרית.

ה"סוד המשותף" הזה צריך להיות באורך מינימלי של 48. והוא מיוצר על ידי הלקוח בעזרת מחולל מספרים פסאודו-אקראיים קריפטוגרפי (Cryptographically secure pseudorandom number generator) או בקיצור CSPRNG, רכיב זה בדרך כלל נמצא במערכת ההפעלה והוא אמור לספק לנו ערכים רנדומילים לצורכי הצפנה.

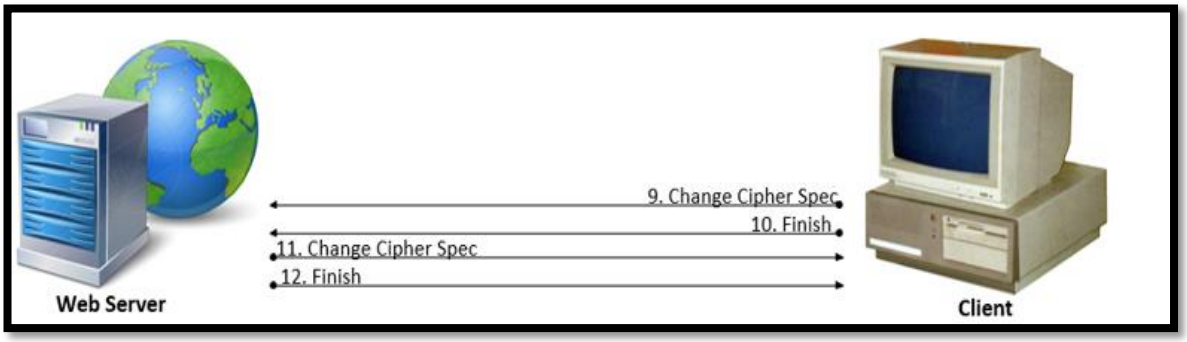
למעשה בעזרת ה-Random ששלחנו את בשלב א' (Client Hello) וה-Pre-MasterSecret השרת והלקוח ייצרו סוד משותף שנקרא "master secret". את הסוד המשותף אנו כמובן נעביר על גבי ההצפנה הא-סימטרית שאותה קבענו ב-cipher suite הראשוני.

הלקוח מחזיר Change_cipher_spec לצורך אישור במעבר לתקשורת בתקשורת סימטרית (כגון AES, DES, 3DES וכו').

ולאחר מכן הודעת ה-finish, אשר מכילה MAC (Message Authentication Code) ו-Hash של הודעות ה-Handshake הקודמות, המודיעה שהתהליך הסתיים בהצלחה מצד הלקוח, השרת מצדו יצטרך לבדוק שה-MAC וה-Hash אכן זהים לאלו שהוא חישב, במידה ולא - השרת יצטרף להרוג את החיבור.

לאחר שהעברנו את ה"סוד המשותף" לשרת - יתחיל שלב ד'. בשלב זה, השרת ישלח לנו בקשת "Change_cipher_spec" שהיא למעשה בקשה למעבר לתקשורת תחת תקשורת סימטרית לפי מה

שבחרנו ב-cipher suite, והודעת finish המודיעה שהתהליך הסתיים בהצלחה מצד השרת. כמו כן הלקוח יצטרף לבצע גם הוא וידוא ל-MAC ול-Hash שקיבל מהשרת.



ובסוף השלב הזה השרת והלקוח מסוגלים לתקשר על גבי תקשורת מוצפנת בהצפנה סימטרית, ללא העברה של המפתח ברשת, וללא סיכום מראש על "סוד משותף".

הרחבה על אלגורתם אוקלידס

אלגוריתם אוקלידס, הוא אלגוריתם אריתמטי המאפשר למצוא, בהינתן שני מספרים טבעיים, את המחלק המשותף המקסימלי שלהם. כפי שלמדנו קודם מחלק המשותף המקסימלי של מספרים, המסומן ע"י gcd(a,b) או בקיצור (a,b). פעולה זו מקבלת שני מספרים טבעיים, ומחזירה את המספר הגדול ביותר שמחלק את שניהם.

לדוגמה, המחלק המשותף המקסימלי של 36 ו-24 הוא 12, מאחר שהמספר מחלק את שניהם ואין מספר גדול יותר בעל תכונה זאת.

דרך אפשרית למציאת המחלק המשותף המקסימלי היא פירוק שני המספרים לגורמים ראשוניים, והכפלת הגורמים המשותפים. בדוגמה הנ"ל, הפירוק לראשוניים של 36 הוא 3×3×2×2 ושל 24 הוא 3×2×2×2, הגורמים המשותפים הם 2, 2 ו-3, ומכפלתם היא 12.

דרך זו היא אינטואיטיבית, אך אינה שימושית עבור מספרים גדולים, כי פירוק לגורמים הוא פעולה מורכבת ואין שיטה פשוטה לחישובו. אלגוריתם אוקלידס מאפשר את מציאת המחלק המשותף המקסימלי ללא פירוק לגורמים, בדרך פשוטה יחסית.

האלגוריתם מבוסס על העיקרון הבא: הוספת כפולה של אחד המספרים למספר השני, אינה משנה את המחלק המשותף הגדול ביותר (מספרים טבעיים בלבד):

$$b, c + qb = (b, c)$$

כלומר עצם זה שהוספנו את q*b לא ישנה את ה-gbc של b ו-c.

סיכום

אנו משתמשים בפרוטוקול זה באופן יום-יומי, אם בעת רכישה באתר אינטרנט, אם בעת גלישה לתיבת הדוא"ל שלנו ואם בעת שיחת סקייפ. החשיבות של הפרוטוקול הנ"ל ברורה מאוד. וכעת, לאחר קריאת המאמר אני גם מקווה שאופן הפעולה שלו בהיר ומובן יותר.

בחלק ב' של המאמר (בתקווה שאספיק לעשות זאת בזמן הקרוב), אמשיך לפרט על המשך ההתקשרות של TLS ועל הצפנות סימטריות, אולי אפילו אכתוב המשך נוסף על ההצפנות הא-סימטריות כגון Elliptic Curves.

השתדלתי להביא את המאמר בשפה פשוטה כדי שיתאים לכל קורא בכל רמת ידע. מקווה שנהנתם ☺.

תודות

תודה לרועי יעקובוביץ, לעידן כהן היקר (ובהצלחה בסטארט-אפ החדש Reflectiz), אסף כץ ודור גרינבאום על העזרה במתמתיקה.

ותודה לכסיף דקל ואפיק קסטיאל על עריכת המאמר.

מקורות מידע

- https://en.wikipedia.org/wiki/Transport_Layer_Security
- https://tools.ietf.org/html/rfc2104
- https://he.wikipedia.org/wiki/%D7%A6%D7%95%D7%A4%D7%9F_%D7%A1%D7%99%D7%9E%D7%98%D7%A8%D7%99
- https://he.wikipedia.org/wiki/%D7%9E%D7%A4%D7%AA%D7%97_%D7%A6%D7%99%D7%91%D7%95%D7%A8%D7%99
- https://he.wikipedia.org/wiki/%D7%A4%D7%A8%D7%95%D7%98%D7%95%D7%A7%D7%95%D7%9C_%D7%93%D7%99%D7%A4%D7%99-%D7%94%D7%9C%D7%9E%D7%9F
- http://www.digitalwhisper.co.il/files/Zines/0x02/DW2-1-SSL.pdf
- https://en.wikipedia.org/wiki/RSA_(cryptosystem)
- http://www.muppetlabs.com/~breadbox/txt/rsa.html