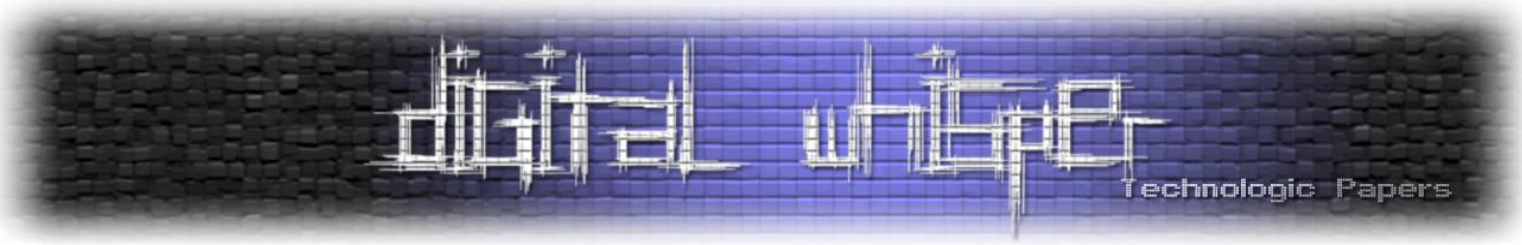


פרוטוקול TLS 1.2

מאת ברק גונן

רקע

הספר "רשתות מחשבים חלק 2" צפוי לצאת בתחילת 2026 בהוצאת המרכז לחינוך סייבר ולהיות נגיש חינם באתר המרכז. לקוראי DigitalWhisper מובא הפרק אודות TLS 1.2.

חלקו הראשון של הספר "רשתות מחשבים" יצא בשנת 2014, הספר הקנה ללומדים את הידע הבסיסי לטובת הבנת עולם רשתות המחשבים. מטרת הספר הייתה לענות על השאלה "כיצד מחשבים מעבירים מידע זה לזה?"

הספר עסק במודל השכבות ופירט את הפרוטוקולים הנפוצים והחשובים, אך נמנע מעיסוק בנושאי הצפנה ואבטחה. בלי אבטחה, האינטרנט לא מסוגל להתקיים, לא ניתן לקיים ברשת פעילות מסחר ועוד. הספר "רשתות מחשבים חלק 2" יענה על השאלה "כיצד מחשבים מעבירים מידע זה לזה בצורה מאובטחת?"

מאז פרסום חלקו הראשון של ספר רשתות השתנו דברים רבים באינטרנט. הסנפת Wireshark שתבוצע כעשור לאחר פרסום ספר רשתות תיראה שונה למדי. השינוי הבולט ביותר: התמעטות עד כדי היעלמות של פרוטוקול HTTP והחלפתו ב-HTTPS, פרוטוקול מאובטח.

גם פרוטוקולים מוכרים לנו עברו שינוי - בספר רשתות הוסבר פרוטוקול HTTP גרסה 1.1, כיום ישנו ל-HTTP גרסה 2 וגרסה 3. פרוטוקול DNS אמנם לא השתנה אבל צורת ההעברה שלו השתנתה. דפדפן ששולח בקשת DNS צפוי להעביר אותה מעל סוקט מאובטח, תוך שימוש בפרוטוקול HTTP, מה שנקרא DoH – DNS over HTTP.

אם לא די בכך, נשברה המוסכמה שתקשורת אמינה עוברת מעל TCP. פרוטוקול QUIC הוא פרוטוקול שלא היה קיים כלל ב-2014, כיום הוא עבר תקינה על ידי גוף בינלאומי ונכנס לשימוש. החידוש - סוקטים מאובטחים ואמינים מעל UDP.

חלקו השני של ספר רשתות מחשבים נכנס אל תוך העולם המרתק של הצפנה. נלמד מהי הצפנה סימטרית ואסימטרית, נלמד פונקציות גיבוב Hash ונבין כיצד בשילוב עם הצפנה ניתן לייצר "חתימה" ייחודית. נבין איך ניתן לייצר אימות באמצעות סרטיפיקט ונראה איך רעיונות מורכבים מתחברים בכל פעם שאנחנו מבצעים גלישת אינטרנט.

סדר הפרקים:

- מבוא לסוקטים מאובטחים
- הצפנה סימטרית
- החלפת מפתחות סימטריים
- Integrity, Authentication
- סרטיפיקטים
- TLS 1.2
- TLS 1.3
- QUIC
- HTTP2, HTTP3, DNS over HTTP

אלפי שנים של צבירת ידע אנושי עובדות מאחורי הקלעים בכל פעם שאנחנו מבצעים גלישת אינטרנט. עולם ידע חדש מחכה לנו, בואו נצלול אליו.

מאמר זה מוקדש לזכרם של סא"ל יצחק הרוש, סמל אורן הרשקו, רס"ן אומרי חי בן משה, סגן ערן שלם, סגן איתן אבנר בן יצחק, סגן רון אריאלי. שמותיהם הותרו לפרסום עם תחילת הכתיבה של פרק זה. מי ייתן ויהיו הנופלים האחרונים.

הקדמה

מטרת הפרק היא להבין את תהליך ה-handshake של פרוטוקול TLS1.2. לפני שנכנס להסבר על גרסה 1.2 נזכיר מה שפירטנו מוקדם יותר על גרסאות ה-TLS. ל-TLS ולקודמו, SSL, יש גרסאות שהוכרזו לא מאובטחות ואינן בשימוש, ושתי גרסאות שעדיין בשימוש: גרסה 1.2 היא הגרסה הוותיקה ביותר, משנת 2008. גרסה 1.3 היא העדכנית יותר, משנת 2018.

פרק זה יעסוק ב-TLS 1.2 הן מכיוון שגרסה זו נמצאת עדיין בשימוש נפוץ והן מכיוון שהבנה שלה היא חשובה ביותר כדי להבין את גרסה 1.3, שיוקדש לה פרק נפרד.

למעשה, כל הפרקים שלמדנו עד כה מטרתם היתה לצקת את בסיס הידע הנדרש כדי שתיאור ה-handshake יהיה פשוט, יחסית. פרק זה יחבר את כל הידע שלמדנו: הצפנות סימטריות, החלפת מפתחות, אלגוריתמי hash, חתימות וכמובן סרטיפיקטים.

נתחיל בהסבר מהו TLS record. לאחר מכן נסקור את ה-TLS handshake בשתי גרסאות- RSA ו-DH. נסיים בהסבר איך נוצרים מפתחות ההצפנה.

כדי להבין איך הכל מתחבר, נעשה שימוש רב ב-Wireshark, ותוך כדי גם נלמד כיצד לפענח הצפנה של הסנפות בעזרת מפתחות.

TLS Records

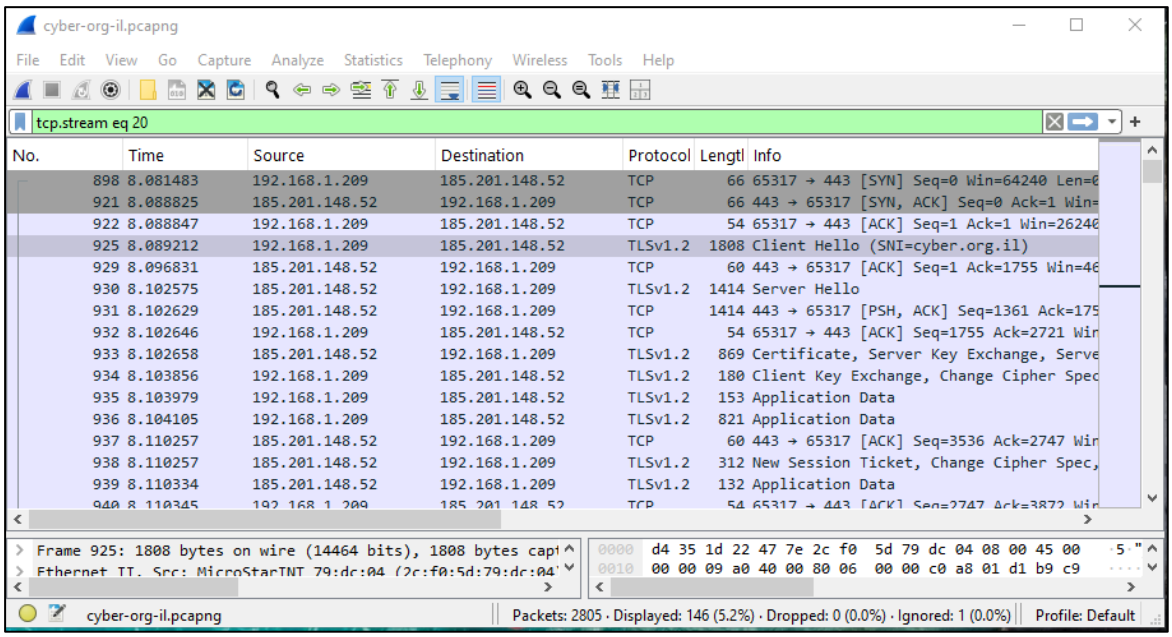
פרוטוקול TLS מחליף רשומות, או records, בין השרת והלקוח. כלומר, השרת והלקוח עדיין מעבירים פקטות זה לזה, אך כל פקטה יכולה להכיל record אחד או יותר. ה-records מאורגנים במבנה שכולל שלושה שדות, ולאחר מכן את ה-records עצמן. שלושת השדות הם:

- Content Type
- Version
- Length

נתחיל בלימוד תוך כדי הסנפה. הורידו את הקובץ הבא, הכולל הסנפה לאתר המרכז לחינוך סייבר:
https://data.cyber.org.il/networks/cyber-org-il-tls.pcapng

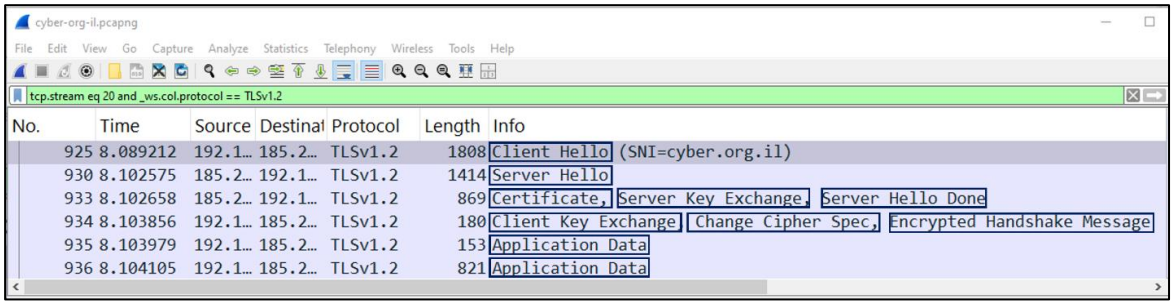
אתם כמובן יכולים לבצע את ההסנפה הזו בכוחות עצמכם, אולם סביר שעם הזמן גרסת ה-TLS של האתר תשתנה, לכן עדיף לעבוד עם קובץ ההסנפה בשלב זה.

באמצעות פילטור, קבלו את זרם הפקטות השייך לתקשורת בין הלקוח לבין cyber.org.il. תזכורת- אפשר להתחיל מפילטר "frame contains "cyber ולהמשיך עם קליק ימני ו-Follow TCP Stream:



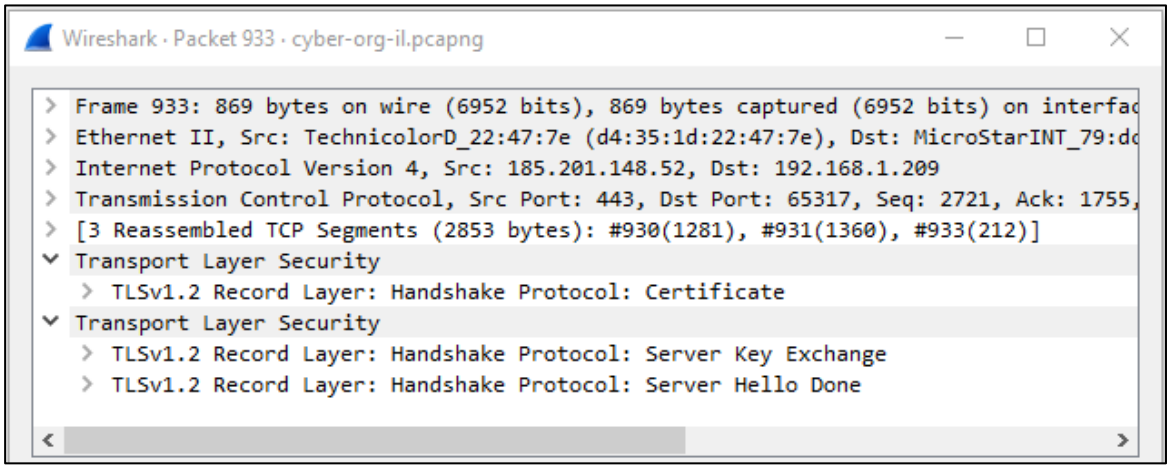
ניתן לראות פקטות משני פרוטוקולים- TCP ו-TLS1.2. שימו לב שהפקטה הראשונה של TLS1.2 נשלחת מהלקוח אל השרת מיד לאחר סיום ה-TCP Three Way Handshake. במילים פשוטות, השרת והלקוח סיימו את לחיצת היד של TCP ומיד מתחילים בהקמת קישור TLS. אין שום דבר אחר שצריך להתרחש בין לבין. שימו לב גם לשוני מהסנפות HTTP שסקרנו בחלקו הראשון של הספר, בהן לאחר ה-Three Way Handshake נשלחה מהלקוח בקשת HTTP.

לאחר הקמת קישור ה-TCP, הפקטות שנשלחות ומסומנות תחת פרוטוקול TCP הן חלקים של TLS1.2. אין לנו עניין בחלקים אלא בצירוף שלהם ל-TLS Records, לכן לטובת הצגה נוחה יותר, נדייק את הפילטר כך שיציג רק פקטות של פרוטוקול TLS1.2:



לטובת ההמחשה, כל אחת מה-Records, מעכשיו נקרא להן "רשומות", מודגשת בריבוע כחול. אפשר לראות שכל התקשורת בין השרת והלקוח מורכבת מרשומות מסוגים שונים, ושלעיתים פקטה אחת של TLS כוללת יותר מאשר רשומה אחת. כעת נסקור את מבנה הרשומות של TLS.

הקליקו על פקטה 933. ניתן לראות שפקטה זו מורכבת משלוש רשומות:



כל רשומה שנבחר, בין אם מפקטה זו או מפקטה אחרת, תכלול את השדות שהצגנו בפתיחה:

```
Transport Layer Security
    TLSv1.2 Record Layer: Handshake Protocol: Certificate
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 2848
        Handshake Protocol: Certificate
Transport Layer Security
    TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 589
        Handshake Protocol: Server Key Exchange
    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
```

Content Type

ישנן רשומות TLS מסוגים שונים. בשלב התחלתי זה אנחנו עדיין בשלב ה-TLS Handshake, שסימונו בפרוטוקול הוא 22, לכן כל הרשומות שלנו הן מסוג 22. אם תקליקו על פקטה 935 לדוגמה, תראו שהרשומה שלה היא מסוג Application Data, שסימונו הוא 23.

המיספור של ה-Content Type לפי הפרוטוקול הוא:

- 20: סוג "Change Cipher Spec". נלמד עליו בקרוב, משמעותו "התחל להצפין"
- 21: סוג "Alert". אזהרה יכולה להיות או Warning או Fatal. נקבל Fatal במקרים כגון כישלון של ה-TLS Handshake, או כאשר הסרטיפיקט פג תוקף.
- 22: כפי שראינו, "Handshake"
- 23: כפי שראינו, "Application Data"

Version

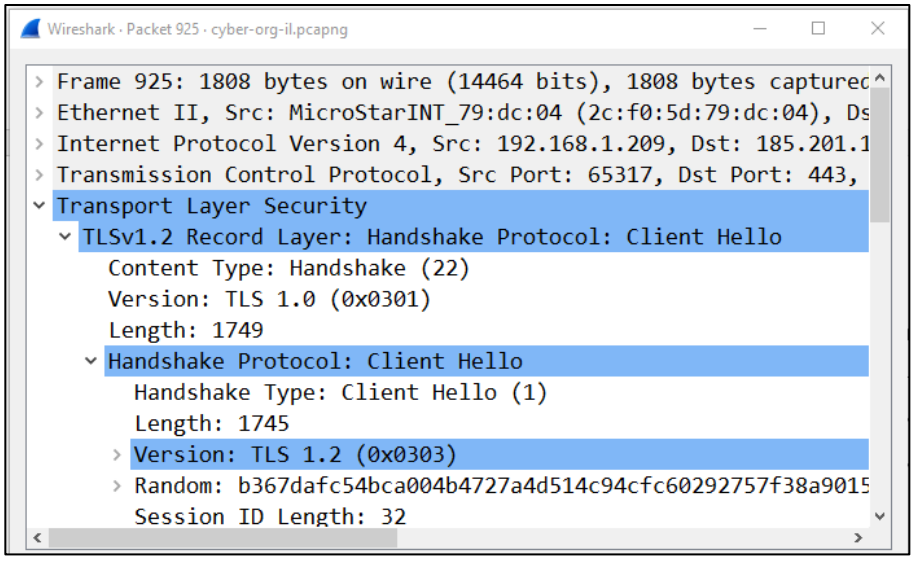
שדה זה מציין את מספר הגרסה של ה-TLS שנעשה בה שימוש:

- 0x0301 - TLS1.0
- 0x0302 - TLS1.1
- 0x0303 - TLS1.2
- 0x0304 - TLS1.3

התקשורת שאנחנו מנתחים כרגע היא TLS1.2 ולכן הערך של השדה הוא 0x0303.

נקודה חשובה - לעיתים קרובות, רישום הגרסה ברשומה אינו מדויק. הסיבה לכך היא שבין השרת והלקוח ישנם רכיבי רשת שייתכן ונוצרו טרם תקופת TLS1.2. רכיבים אלו עלולים להפיל פקטות שגרסת ה-TLS שלהן חדשה. לכן, התקשורת בין השרת והלקוח "מתחזה" לגרסה נמוכה יותר.

הבה ניווכח בתופעה הזו. פיתחו את פקטה 925:



כפי שרואים, בשדה ה-Version של הרשומה נכתב TLS 1.0. עם זאת, בדיקה מעמיקה יותר לתוך המידע שעובר מתחת, מראה כי אכן מדובר בגרסה 1.2. על שדה האורך אין מה לפרט, בשלב זה אתם כבר מכירים איך הוא עובד. לאחר מכן יופיע המידע עצמו שעובר ברשומה.

כעת, לאחר שהבנו כי השרת והלקוח מחליפים ביניהם רשומות TLS, נוכל לעבור על ה-Handshake.

בסעיף זה אנחנו קוצרים את הפירות על כל ההכנה הארוכה שעשינו בפרקים הקודמים. כל מה שלמדנו עד כה נועד להביא אותנו לנקודה שבה נוכל להבין את הפסקאות הבאות.

תיאום Cipher Suites

בפרקים הקודמים ראינו שכדי לקיים את עקרון ה-Confidentiality, Integrity, Authentication השרת והלקוח נדרשים לכמה כלים. ראשית הם נדרשים לאלגוריתם הצפנה סימטרי. אלגוריתם זה יכול להיות לדוגמה AES. האלגוריתם עצמו אינו מספיק, נדרש גם מפתח הצפנה משותף. כדי לתאם את מפתחות ההצפנה, נדרש אלגוריתם החלפת מפתחות. אלגוריתם זה יכול להיות RSA או DH. הבחירה באלגוריתם החלפת המפתחות קובעת מי שולח למי מה. אם נבחר DH, כל צד ישלח לצד השני את החלק הגלוי שלו ביצירת הסוד המשותף. אם נבחר RSA, צד הלקוח יבחר סוד ויצפין אותו באמצעות המפתח הציבורי של השרת. אם השרת והלקוח תיאמו את האלגוריתמים הללו ביניהם, יש להם Confidentiality.

כדי שהלקוח יבטח בסרטיפיקט של השרת, וכך יתקיים ביניהם Authentication, השרת והלקוח צריכים לתאם ביניהם אלגוריתם חתימה. אלגוריתם זה עשוי להיות RSA, או DSA. אלגוריתם DSA הוא קיצור של Digital Signature Algorithm. מבלי להכנס לפרטים של DSA, אפשר להבין משמו שהוא עשוי לשמש כחלופה ל-RSA לטובת המלאכה של חתימה דיגיטלית.

השרת והלקוח צריכים גם להבטיח Integrity, שאף גורם לא ישנה את הפקטות שעוברות ביניהם. הם מבצעים זאת באמצעות הוספת HMAC, Hashed Message Authentication Code, כך שנדרש לתאם ביניהם גם אלגוריתם Hash.

אם כך, לפני שהשרת והלקוח יכולים להקים סוקט מאובטח הם צריכים לתאם ביניהם את ארבעת הדברים הללו:

1. מהו האלגוריתם שישמש להחלפת מפתחות ההצפנה הסימטריים?
2. מהו אלגוריתם החתימה?
3. מהו אלגוריתם ההצפנה סימטרי?
4. מהו אלגוריתם ה-Hash?

התשובה לארבעת השאלות הללו היא מה שנקרא ה-**Cipher Suite** שהשרת והלקוח בחרו. לדוגמה, הצירוף DH-RSA-AES256-SHA256 הוא צירוף של רביעיית אלגוריתמים שעשוי להבחר בתור Cipher Suite.

יצירת Master Secret

בנוסף לבחירת ארבעת החלקים של ה-Cipher Suite, השרת והלקוח צריכים להחליף גם ביניהם את כל המידע לטובת יצירת המפתחות שישמשו אותם בהמשך. לטובת הדיון על יצירת מפתחות חשוב להזכר, שכאשר דנו בסוקטים בשכבת התעבורה, ראינו שסוקט מורכב משני זרמים של מידע, זרמים שאינם תלויים זה בזה. לדוגמה, כאשר לקוח פותח סוקט TCP מול השרת הוא בוחר מספר SEQ משלו, ול-SEQ של השרת אין קשר ל-SEQ של הלקוח. באותו האופן, הכיוון של הסוקט שבין הלקוח והשרת מאובטח על ידי מפתחות שונים מאשר הכיוון שבין השרת והלקוח. אם כך אנחנו מגיעים למסקנה שנדרשים שני מפתחות הצפנה סימטריים, אחד לכל צד.

כדי שכל פקטה שנשלחת תכלול HMAC, שיעיד על ה-Integrity שלה, צריך גם מפתח שישמש את הצדדים. נזכיר כי HMAC פועל בדרך הבאה: לוקחים מפתח סודי שתואם בין הצדדים, ומחברים אותו למסר שמעוניינים להעביר. מבצעים Hash על התוצאה, ומקבלים HMAC. שולחים את המסר המקורי - ללא המפתח כמובן - יחד עם ה-HMAC.

כלומר סוקט מאובטח צריך ארבעה מפתחות:

- מפתח הצפנה סימטרי לצד השרת
- מפתח הצפנה סימטרי לצד הלקוח
- מפתח HMAC לצד השרת
- מפתח HMAC לצד הלקוח

עקרונית, השרת והלקוח יכלו להשתמש באלגוריתם החלפת המפתחות (RSA או DH) כדי להעביר כל אחד מארבעת המפתחות. מעשית, כדי לייעל את התהליך, השרת והלקוח מתאמים ביניהם מפתח יחיד, שבדרך מסויימת "מתחלק" לארבעת המפתחות. התיאום של אותו מפתח יחיד, שנקרא **Master Secret**, הוא המטרה הנוספת של ה-TLS Handshake.

ביצוע Authentication לצד השרת

כחלק מתהליך ה-TLS Handshake, הלקוח צריך לוודא שהשרת הוא אכן מי שהוא נחזה להיות. פעולה זאת מתבצעת באמצעות הסרטיפיקט של השרת, אותו הוא מעביר ללקוח. הלקוח צריך גם לוודא שהשרת הוא בעל המפתח הפרטי שמתאים לסרטיפיקט שהוצג לו. לכן, הלקוח והשרת משתמשים במפתח הציבורי שכתוב בסרטיפיקט כחלק מהתהליך תאום המפתח ה-Master Secret.

מניעת MITM

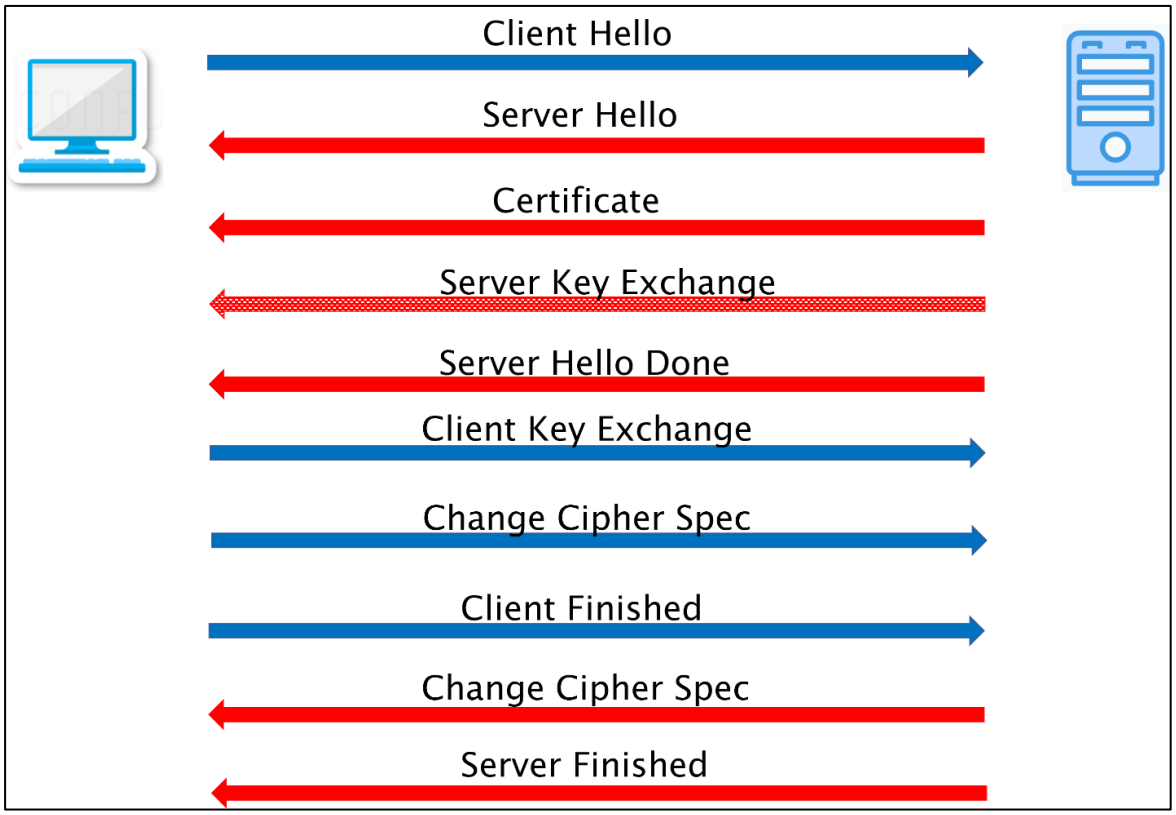
השרת והלקוח ערים לכך שגורם זדוני כלשהו עלול לבצע ביניהם מתקפת Man In The Middle בזמן ה-Handshake. מתקפה כזו תכלול מסרים מהונדסים כך שבסופו של דבר כל צד יפתח סוקט מאובטח מול הגורם הזדוני, שישיג את מפתחות ההצפנה ואת ה-HMAC של שני הצדדים. לאחר מכן הגורם הזדוני יתווך בין השרת

והלקוח ויוכל למעשה לקרוא ואף לשנות את המידע שעובר ביניהם. לכן, הצדדים צריכים לוודא שכל צד לתקשורת קיבל בדיוק את מה שהצד השני שלח, לכל משך ה-Handshake, ללא שינוי.

Handshake - בגרסת DH

כפי שתארנו, אחד מתפקידי ה-TLS Handshake הוא לתאם את ה-Master Secret, הסוד המשותף שממנו ייגזרו כל המפתחות. לכן הגיוני שה- Handshake משתנה לפי הדרך הנבחרת להחלפת מפתחות סימטריים.

האיור הבא מתאר את רשומות ה-TLS המוחלפות בין השרת והלקוח כאשר אלגוריתם החלפת המפתחות שנבחר הוא DH. אחת הרשומות אינה קיימת בגרסת RSA של ה-Handshake, לכן היא מסומנת בצבע שונה.



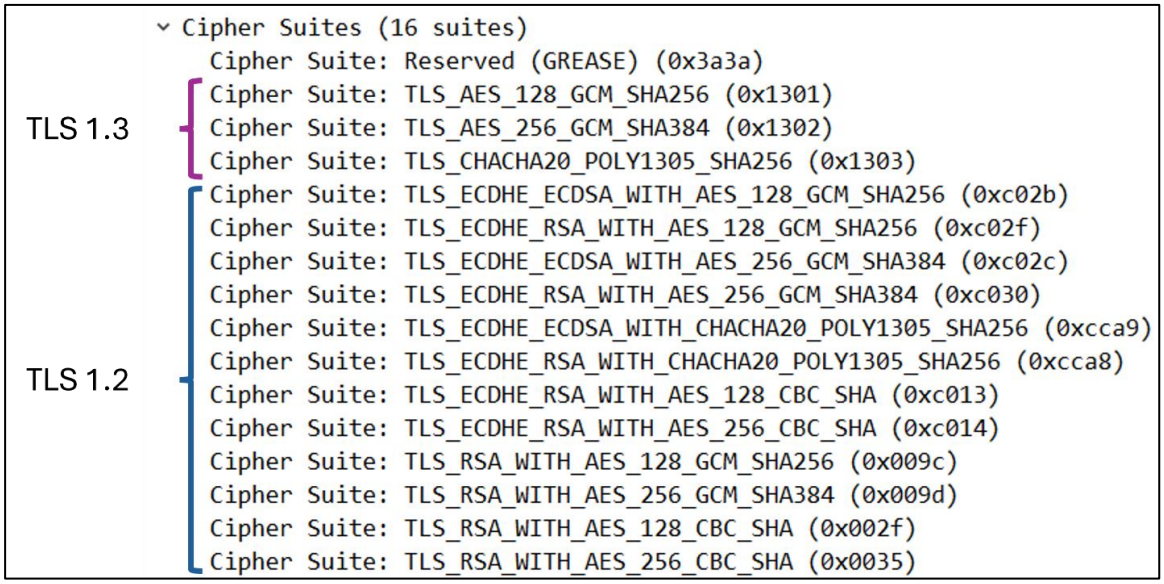
Client Hello

פיתחו את פקטה 925 בקובץ ההסנפה, הכוללת את הרשומה של Client Hello.

תחילת הרשומה כוללת מספר שדות שאנחנו כבר מכירים ולכן נעבור עליהם בקצרה. הרשומה מתחילה בשלושת השדות של סוג, גרסה ואורך, אותם סקרנו בתחילת הפרק. הגרסה היא TLS 1.0, כזכור המטרה היא רק לעבור בשלום רכיבים ישנים שאולי נמצאים בין השרת והלקוח. לאחר מכן מתחילה הרשומה עצמה, מסוג Handshake. כיוון שלתהליך ה-Handshake עצמו יש סוגים רבים של רשומות, אותן מיד נכיר, יש צורך לציין

שזוהי רשומה מסוג Client Hello, שסימונה בפרוטוקול "1". לאחר מכן יש שדה אורך נוסף וגרסה - הפעם, הגרסה הנכונה TLS 1.2.

עיברו לשדה ה-Cipher Suites. הרחיבו את התצוגה כדי לראות את הפרטים:



הלקוח מצהיר על כל האפשרויות שהוא תומך בהן. במקרה זה הלקוח מסר לשרת שהוא תומך ב-16 אפשרויות. האפשרות הראשונה שמורה לניסיונות וחידושים בתקן, לא רלבנטית. שלושת האפשרויות הבאות שייכות לגרסת TLS 1.3, נדון בהן בהמשך.

האפשרויות שנותרו הן מהצורה הבאה:

"TLS_X_Y_WITH_Z_W"

כאשר במקום X תבוא שיטת החלפת המפתחות הסימטריים.

Y- אלגוריתם החתימה.

Z - אלגוריתם ההצפנה הסימטרית.

W - אלגוריתם ה-Hash.

לדוגמה:

TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

החלפת מפתחות סימטריים - DH (אם לדייק, ECDHE, קיצור של Elliptic Curve Diffie Hellman Ephemeris, וריאציה של דיפי הלמן)

חתימה - RSA.

הצפנה סימטרית - AES 128 GCM. כזכור זוהי וריאציה של קוד הבלוק AES, שבה הבלוק הוא בגודל 128 ביט ונעשה שימוש ב-Counter Mode, כלומר כל בלוק מוצפן עם מפתח הכולל מספר סידורי, ייחודי לבלוק.

אלגוריתם Hash - SHA256.

יש כמה Cipher Suites חריגים. ארבעת האחרונים חורגים מהמבנה שהוצג, ולכאורה הם כוללים רק שלושה אלגוריתמים. לדוגמה:

TLS_**RSA**_WITH_AES_128_GCM_SHA256

הסיבה לכך היא ש-RSA הוא ייחודי, בכך שיכול לשמש הן כאלגוריתם החלפת מפתחות והן כאלגוריתם חתימה. במקרה ש-RSA כתוב כך, במבנה של שלשת אלגוריתמים, הכוונה היא ש-RSA משמש בתפקיד כפול. עד כאן אודות ה-Cipher Suites.

Server Hello

השרת קיבל מהלקוח פניה, Client Hello, והוצעו לו מספר Cipher Suites לבחור מהם.

השרת בוחר לעבוד בגרסת TLS 1.2, ומודיע ללקוח מהו ה-Cipher Suite שהוא בחר:

```
▾ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 70
    Version: TLS 1.2 (0x0303)
  › Random: 2a4310376d0f8b53d2eb78cbff686826f271dcd86b13cccee0c8d345732a1c8d
    Session ID Length: 0
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
```

בהמשך נתייחס לשדות נוספים שעוברים ב-Server Hello.

Certificates

ברשומה זו השרת מעביר ללקוח את הסרטיפיקט שלו, יחד עם סרטיפיקט של ה-ICA שחתם עליו (בהנחה שהוא לא נחתם ישירות על ידי ה-Root CA).

ניתן לראות שני סרטיפיקטים:

```
▾ Handshake Protocol: Certificate
    Handshake Type: Certificate (11)
    Length: 2844
    Certificates Length: 2841
  ▾ Certificates (2841 bytes)
      Certificate Length: 1545
    › Certificate […]: 30820605308204eda0
      Certificate Length: 1290
    › Certificate […]: 30820506308202eea0
```

נפתח את הסרטיפיקט הראשון ונמצא מי ה-Issuer שלו ולמי הוא שייך, כלומר מי נמצא בשדה ה-Subject שלו:

```
▾ Certificate […]: 30820605308204eda00302010202120 60fe9e33f58
  ▾ signedCertificate
      version: v3 (2)
      serialNumber: 0x060fe9e33f58be8d30bc4f7845c787218a1f
    › signature (sha256WithRSAEncryption)
    ▾ issuer: rdnSequence (0)
      › rdnSequence: 3 items (id-at-commonName=R12,id-at-orga
    › validity
    ▾ subject: rdnSequence (0)
      › rdnSequence: 1 item (id-at-commonName=cyber.org.il)
    › subjectPublicKeyInfo
```

כפי שרואים, R12, שהוא כזכור ICA, חתם לדומיין cyber.org.il.

פיתחו את הסרטיפיקט השני ותמצאו שם את הסרטיפיקט של 12R, חתום על ידי ה-Root CA.

Server Key Exchange

ההסבר על רשומה זו הוא דוגמה קלאסית לכך שהודות לבסיס שבנינו אנחנו יכולים להבין בקצרה פעולה מורכבת.

ברשומה הקודמת, השרת הכריז על בחירה באלגוריתם DH לטובת יצירת הסוד המשותף, הוא ה-Master Secret, שממנו כאמור ייגזרו הן מפתחות ההצפנה הסימטריים והן ה-HMAC-ים.

שום דבר לא מעכב את השרת כבר להתחיל בתהליך ולשלוח ללקוח את ה-Public Key הנדרש לטובת יצירת הסוד המשותף:

```
▾ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 589
  ▾ Handshake Protocol: Server Key Exchange
      Handshake Type: Server Key Exchange (12)
      Length: 585
    ▾ EC Diffie-Hellman Server Params
        Curve Type: named_curve (0x03)
        Named Curve: secp256r1 (0x0017)
        Pubkey Length: 65
        Pubkey: 04d6136caa0068fb47f7dad1188f377133904a4347ce82a664abe
      ▾ Signature Algorithm: rsa_pkcs1_sha512 (0x0601)
          Signature Hash Algorithm Hash: SHA512 (6)
          Signature Hash Algorithm Signature: RSA (1)
        Signature Length: 512
        Signature […]: 158b250af81cc6a29db74d09c29852acb6e1fbc4dffe81
```

ניתן לראות תחת "EC Diffie-Hellman Server Params" את הערך של ה-Pubkey שיצר השרת. הלקוח יוכל עכשיו לחשב באמצעותו את הסוד המשותף.

כפי שרואים בהמשך השרת מצרף גם חתימה. חישבו, מדוע יש צורך גם בחתימה?

ניזכר במה שלמדנו על סרטיפיקטים. הלקוח חייב לוודא שהשרת הוא אכן הבעלים האמיתי של הסרטיפיקט שנשלח אליו (הרי כל אחד יכול להוריד סרטיפיקט ולשלוח אותו) והדרך שלו לעשות זאת היא באמצעות וידוא שהשרת הוא הבעלים של המפתח הפרטי הצמוד למפתח הציבורי שבסרטיפיקט. השרת מוכיח ללקוח את הבעלות על המפתח הפרטי באמצעות שליחת חתימה על ה-Pubkey. הלקוח ישתמש במפתח הציבורי של השרת, מתוך הסרטיפיקט, כדי לפתוח את החתימה ולוודא שה- Hash מתאים לזה של ה-Pubkey.

והנה שאלה נוספת למחשבה: כזכור, הסרטיפיקט כולל פירוט של אלגוריתמי החתימה וה- Hash שנעשה בהם שימוש לטובת החתימה. מדוע יש צורך שהשרת יציין מהם אלגוריתמי החתימה וה- Hash (במקרה זה RSA מעל SHA512) גם ברשומה זו? האם אין כאן כפילות?

בסרטיפיקט, אלגוריתמי החתימה וה-Hash מתייחסים לדרך שבה ה-ICA חתם על הסרטיפיקט של השרת, ולא לאלגוריתמים שה*שרת* משתמש בהם בשביל לחתום. תוכלו לוודא זאת אם תכנסו לסרטיפיקט של השרת.

- ה-ICA שחתם ל-cyber.org.il השתמש ב-RSA עם SHA256.
- השרת של cyber.org.il השתמש ב-RSA עם SHA512.

רגע אחד! מה קורה כאן? ה-Cipher Suite שהשרת בחר בו היה:

ECDHE_RSA_WITH_AES_128_GCM_SHA256

אם השרת סיכם עם הלקוח על SHA256, מדוע הוא מודיע ללקוח על SHA512?

מסיבות התלויות בגרסאות קודמות של TLS, הערכים שסוכמו ב-Cipher Suite משמשים את כלל התעבורה בין השרת והלקוח, למעט החתימה על ה-Pubkey. ב-Client Hello, בשדה ה-Signature Algorithms, הלקוח הציע מספר אפשרויות של שילובים של Hash וחתימה שהוא תומך בהם. השרת מודיע כאן באיזו אפשרות הוא בחר.

Server Hello Done

כל המשמעות של רשומה קצרה היא "לקוח יקר, אני סיימתי לשלוח אליך כל מה שהייתי צריך לשלוח, בשלב זה. הכדור אצלך":

```
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
```

Client Key Exchange

הלקוח יודע בשלב זה מה ה-Cipher Suite שהשרת בחר, במקרה זה DH. הלקוח קיבל את החלק של השרת ביצירת הסוד המשותף, ה-Master Secret, וכעת הוא עונה עם Pubkey משלו:

```
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: 0463936e928333ee1ec381b8bc36c8e9197205d263e190a4
```

מאחרי הקלעים, הלקוח גם וידא שהשרת הוא אכן הבעלים של המפתח הציבורי בסרטיפיקט.

Change Cipher Spec

בשלב זה גם השרת וגם הלקוח אמורים להיות מסוגלים לחשב את אותו Master Secret. כעת הלקוח שולח הודעה שמשמעותה "עבור למוצפן".

```
TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
```

שימו לב שרשומה זו היא בעלת Content Type שאינו Handshake, כפי שהיו כל הרשומות עד כה. הסיבה לכך היא שרשומה זו עשויה להשלח גם תוך כדי תעבורת תקשורת לאחר ה-Handshake, אם אחד הצדדים מבקש להחליף מפתחות הצפנה.

פענוח תקשורת מוצפנת

אם נפתח את הרשומה נמצא שכתוב שם שזוהי רשומה מוצפנת, בלי פירוט:

```
TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 40
    Handshake Protocol: Encrypted Handshake Message
```

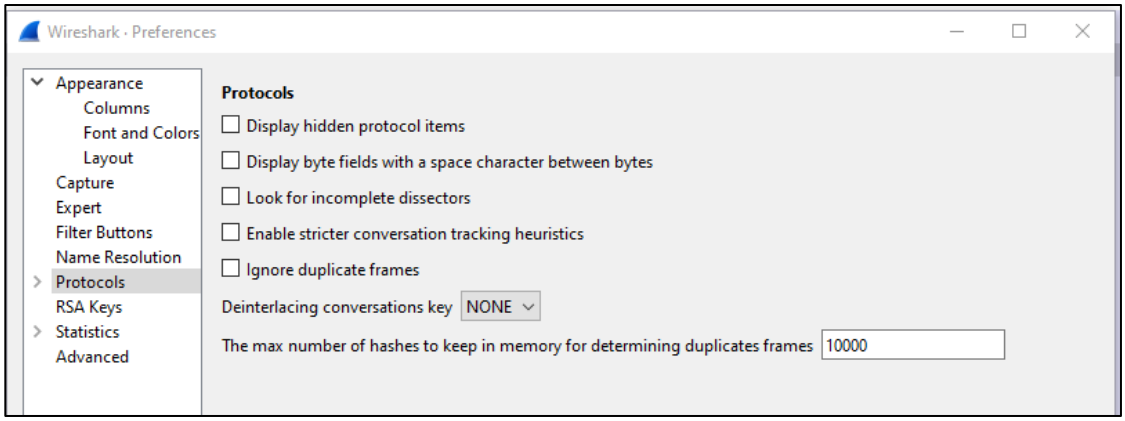
למזלנו, Wireshark יודע לקבל בתור קלט קובץ מפתחות, הכולל את ה-Master Secret של התעבורה, ולהשתמש בו כדי לפתוח את ההצפנה של כל הרשומות. הורידו את הקובץ הבא:

https://data.cyber.org.il/networks/cyber-org-il-tlskeys.txt

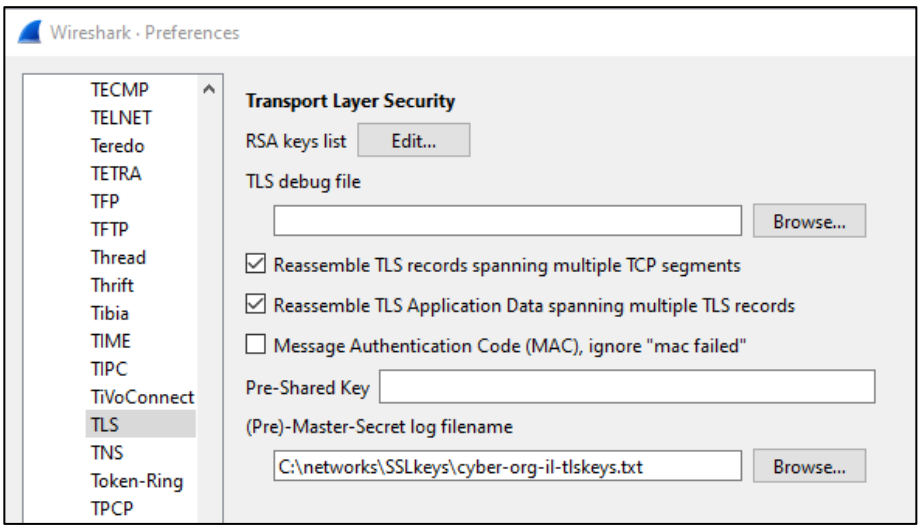
שימרו את הקובץ בתיקיה לבחירתכם. לדוגמה:

c:\networks\SSLkeys

כדי לייבא את הקובץ אל Wireshark, הכנסו לתפריט Edit ובתוכו Preferences:



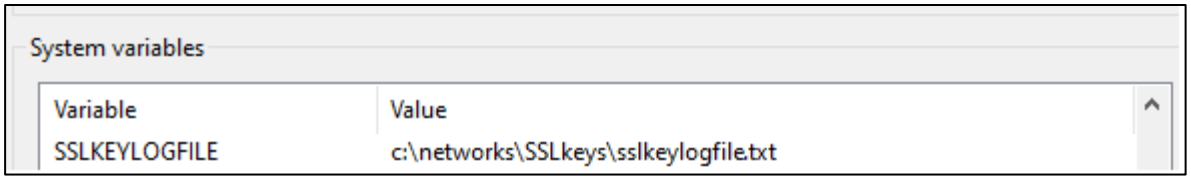
שם תחת Protocols חפשו TLS (אם תכתבו את האות t התפריט יקפוץ לשם מיד ותחסכו זמן גלילה). בתוך TLS, הגדירו היכן נמצא קובץ המפתחות:



לאחר שתקליקו על OK, נכונה לכם הפתעה קטנה - שם הרשומה האחרונה השתנה מ Encrypted Handshake Message ל-Client Finished.

כדי ליצור קובץ מפתחות משלכם, פעלו לפי ההוראות הבאות:

1. בתפריט החיפוש של Windows כיתבו env, והקליקו על Edit the system environment variables
2. בתוך חלון ה System Properties הקליקו על Environment Variables
3. הוסיפו משתנה סביבה חדש בשם SSLKEYLOGFILE. ערכו של המשתנה יהיה שם הקובץ אליו תרצו לשמור את המפתחות, כולל הנתיב המלא
4. בצעו אתחול ולמחשב ותוכלו לוודא שהקובץ נוצר
5. שימו לב - הקובץ ישמור את כל המפתחות הגלישה שלכם מעתה ואילך. אחרי זמן מה הקובץ עלול לגדול מאד, מה שיגרום לכך שייקח ל-Wireshark זמן רב למצוא את המפתחות שהוא צריך. לכן מומלץ אחת לזמן מה למחוק את הקובץ וליצור במקומו חדש. כיוון שהקובץ תפוס על ידי מערכת ההפעלה לא תוכלו למחוק אותו סתם כך, תצטרכו לשנות את משתנה הסביבה SSLKEYLOGFILE לערך חדש (שם קובץ אחר), לאתחל את המחשב, ורק אז תוכלו למחוק את קובץ המפתחות הישן.



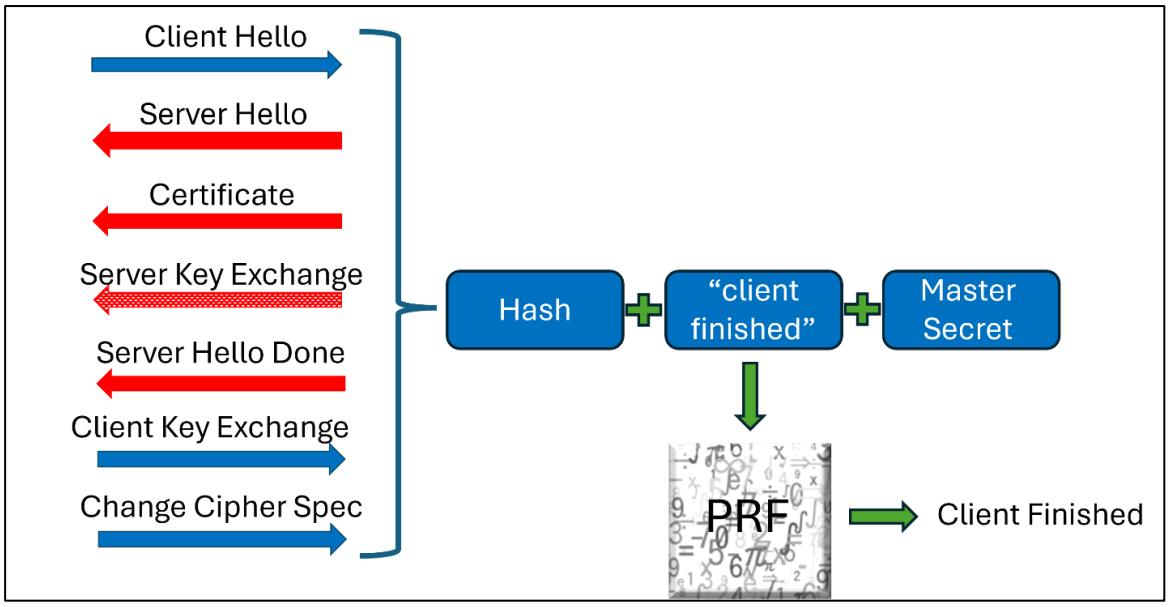
Client Finished

בפתיחת ההסבר על ה-Handshake ראינו שאחת מהמטרות היא מניעת MITM. כלומר, לוודא ששום גורם לא נמצא בין השרת והלקוח ומשנה את ההודעות ביניהם. השרת והלקוח רוצים לוודא שכל ההודעות שלהם התקבלו בדיוק כפי שהן על ידי הצד השני.

ברשומת ה-Client Finished הלקוח מוסר לשרת את המידע שנדרש כדי שהשרת יבדוק אם ההודעות שלו התקבלו על ידי הלקוח בלא שינוי.

הלקוח צובר את כל הרשומות שעברו עד כה בינו לבין השרת ומבצע לצבר הרשומות Hash. ה-Hash מחובר למחרוזת "client finished" ול- Master Secret שהלקוח חישב, ושאמור כמובן להיות זהה בינו לבין השרת. חיבור כל המחרוזות הללו עובר פונקציית ערבול נוספת שנקראת PRF והתוצאה היא ה-Client Finished. אלגוריתם ה-Hash שנעשה בו שימוש הוא אותו אלגוריתם Hash שסוכם ב-Cipher Suites.

לטובת שלמות ההסבר, הנה פסקה אודות PRF. ניתן לדלג, או לצאת למסע חיפוש אינטרנטי, כרצונכם. PRF היא פונקציה המייצרת מספרים "כביכול אקראיים". מחשבים לא באמת מסוגלים לייצר מספרים אקראיים, כיוון שכל החישובים המתמטיים שלהם ניתנים לשחזור. לכן כל הפונקציות שאמורות ליצור מספרים אקראיים הן למעשה Pseudo Random Functions, או בקיצור PRF. הפונקציות הללו מסוגלות לייצר מספרים שנראים אקראיים לכאורה, אבל למעשה הם תלויים בערך התחלתי שהוזן לפונקציה. ערך זה נקרא Seed. מה שהלקוח עושה הוא לחשב Seed שמבוסס על שלושת החלקים שנסקרו, ומעביר אותו ל-PRF, שמייצרת מספר אקראי כביכול, המועבר לשרת.



Change Cipher Spec, Server Finished

נדלג לעת עתה על הרשומה ששלח השרת, New Session Ticket. רשומה זו אינה הכרחית בתהליך ה-Handshake.

השרת עונה ללקוח ב-Change Cipher Spec מצידו, שאומר "גם אני מתחיל להצפין את התקשורת ממני אליך". הודעה זו זהה להודעה שהלקוח שלח.

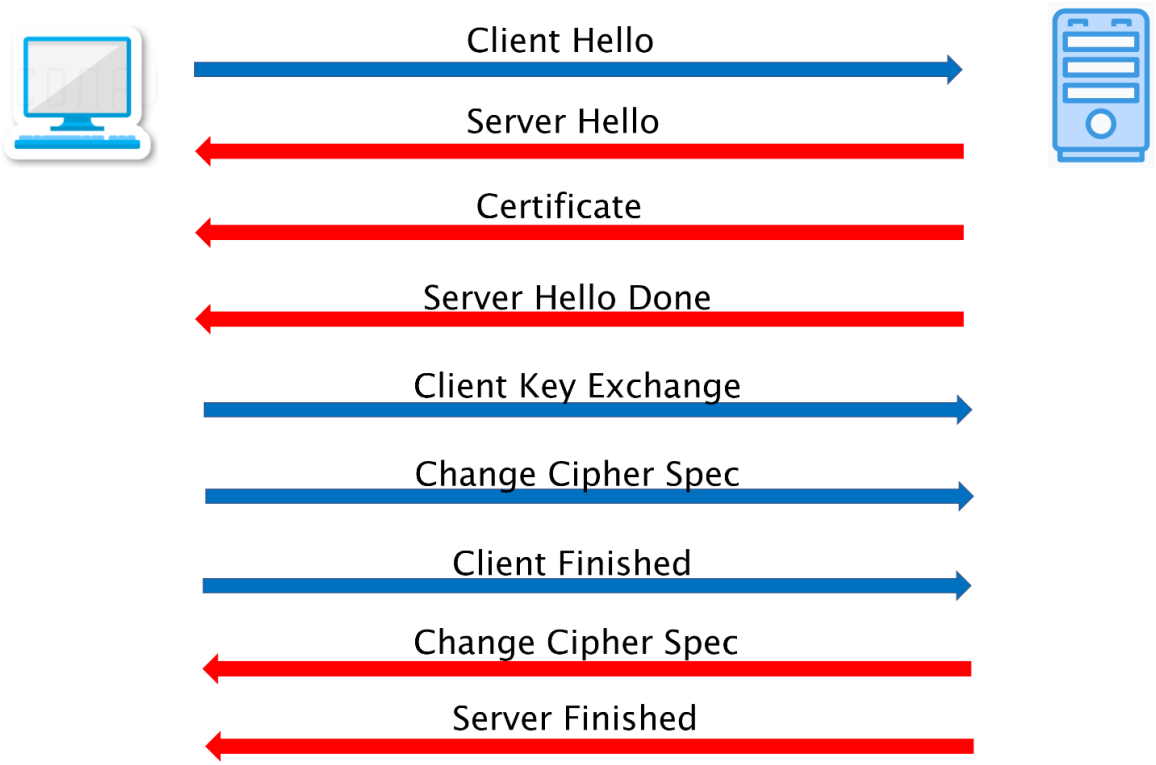
רשומת ה-Server Finished מיועדת לאפשר, הפעם ללקוח, לבדוק שאין MITM בינו לבין השרת. השרת שולח תקציר חתום של כל ההודעות בדומה למה שהלקוח שלח ב-Client Finished, אך עם שני הבדלים קלים:

- הרשומות שעליהן השרת מבצע Hash כוללות את הרשומות שנשלחו מאז ה-Client Finished ועד כה
- המחרוזת שהשרת מוסיף ל-Hash היא מן הסתם "Server Finished"

Handshake - בגרסת RSA

התמקדנו ב-Handshake בגרסת DH מכיוון שהוא נפוץ יותר. די קשה למצוא שרת שיחליף מפתחות באמצעות RSA. למעשה בגרסה הבאה של TLS, גרסה 1.3 האפשרות להחליף מפתחות באמצעות RSA כבר לא קיימת, היא נחשבת פחות בטוחה. עם זאת לטובת שלמות ההסבר, הנה סקירה של RSA Handshake.

החלפת מפתחות בגרסת RSA כוללת את כל הרשומות שאנחנו מכירים, אך אין צורך ברשומה של Server Key Exchange. הסיבה לכך היא שמי שבוחר את הסוד המשותף הוא רק הלקוח. ברשומת ה- Client Key Exchange הלקוח ישלח לשרת את הסוד כשהוא מוצפן באמצעות המפתח הציבורי של השרת.



הפעולה הזו, כפי שכבר תיארנו בפרק על סרטיפיקטים, לא רק מחליפה את המפתח של הלקוח עם השרת אלא גם מאפשרת ללקוח לבדוק שהשרת הוא אכן הבעלים של המפתח הפרטי המתאים לסרטיפיקט. רק הבעלים של המפתח הפרטי יצליח לחלץ את הסוד שהלקוח בחר.

תרגיל מודרך פילטור RSA Handshake

הורידו את קובץ ההסנפה הבא:

https://data.cyber.org.il/networks/RSA_handshake.pcapng

המשימה שלכם היא לפלטר רק TLS Handshake שבהם נעשה שימוש ב-RSA כשיטת החלפת מפתחות.

חישבו - איך עושים זאת?

נזכור, שמי שבוחר את שיטת החלפת המפתחות הוא השרת. המקום בו השרת מכריז על הבחירה שלו היא רשומת ה-Server Hello. לכן עלינו לחפש פקטות שיש בהן Server Hello. הכנסו ל-Server Hello כלשהו ומיצאו איזה שדה קובע את סוג הרשומה.

התשובה:

שדה ה-type בתוך tls.handshake. קיבעו את הערך המתאים בשביל לפלטר Server Hello. אם הצלחתם, קבלתם רשימה של כל רשומות ה-Server Hello. כעת, איך אפשר לפלטר רק רשומות בהן השרת בחר את RSA כאלגוריתם החלפת מפתחות?

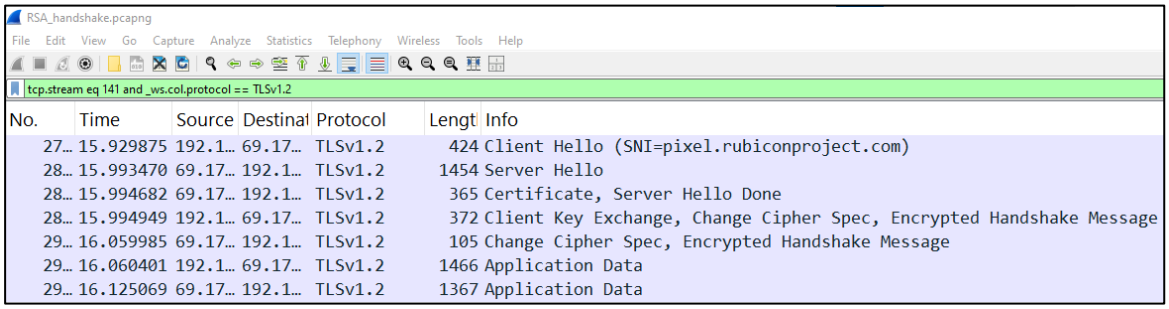
נזכור כי כל ה-Cipher Suites נמצאים ב-Client Hello, כולל המזהים המספריים שלהם בתקן TLS. פתיחת ה-Cipher Suites מראה לנו כי RSA משמש להחלפת מפתחות בארבע אפשרויות, המסומנות בערכים 0x002f, 0x0035, 0x009c, 0x009d.

כעת דייקו את הפילטר שלכם והגיעו אל ה-Handshake הנדרש.

תשובה:

```
tls.handshake.type == 2 and (tls.handshake.ciphersuite == 0x002f or
tls.handshake.ciphersuite == 0x0035 or tls.handshake.ciphersuite == 0x009c or
tls.handshake.ciphersuite == 0x009d)
```

לאחר שמצאתם את הפקטה עם ה-Server Hello, בצעו Follow TCP stream והוסיפו פילטור לפי סוג פרוטוקול, TLSv1.2. התוצאה:



כפי שרואים, אין רשומה של Server Key Exchange.

פיתחו את רשומת ה-Client Key Exchange, ודאו כי הסוד המשותף מוצפן באמצעות מפתח RSA:

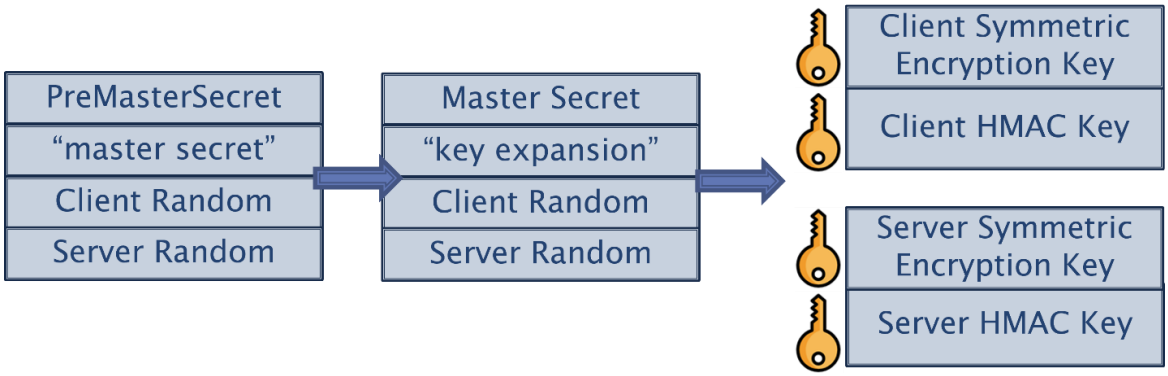
```
▾ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 262
  ▾ Handshake Protocol: Client Key Exchange
      Handshake Type: Client Key Exchange (16)
      Length: 258
    › RSA Encrypted PreMaster Secret
```

שלבי יצירת מפתחות

במהלך סקירת ה-Handshake למדנו שהשרת והלקוח מנסים לתאם ביניהם מפתח שנקרא Master Secret, ושממנו נגזרים יתר המפתחות. כעת נתעמק בתהליך.

הסוד המשותף שהשרת והלקוח מתאמים ביניהם, בין אם באמצעות DH או RSA, נקרא Pre Master Secret. הזכרו בכך, שכאשר ביצענו יבוא של קובץ מפתחות לתוך Wireshark, התבקשנו להזין שם קובץ שכולל Pre Master Secret.

כלומר נקודת ההתפחה של תהליך יצירת המפתחות היא שהצדדים תיאמו Pre Master Secret.



אל ה-Pre Master Secret מתווספת המחרוזת "master secret" ועוד שני מספרים. כאשר סקרנו את הרשומות הללו דילגנו על ה-Random, כעת אנחנו חוזרים למבט נוסף. פיתחו את ה-Client Hello ואת ה-Server Hello של הסנפת cyber.org.il ומצאו אותם:

```
Handshake Protocol: Client Hello
  Handshake Type: Client Hello (1)
  Length: 1745
  Version: TLS 1.2 (0x0303)
  Random: b367dafc54bca004b4727a4d514c94cfc60292757f38a9015b0b52617d7822ae
    GMT Unix Time: May 19, 2065 06:05:32.000000000 Jerusalem Daylight Time
    Random Bytes: 54bca004b4727a4d514c94cfc60292757f38a9015b0b52617d7822ae
```

```
Handshake Protocol: Server Hello
  Handshake Type: Server Hello (2)
  Length: 70
  Version: TLS 1.2 (0x0303)
  Random: 2a4310376d0f8b53d2eb78cbff686826f271dcd86b13cccee0c8d345732a1c8d
    GMT Unix Time: Jun 20, 1992 14:02:15.000000000 Jerusalem Daylight Time
    Random Bytes: 6d0f8b53d2eb78cbff686826f271dcd86b13cccee0c8d345732a1c8d
```

לפי התקן, ארבעת הבתים הראשונים של ה-Random אמורים להיות מבוססים על התאריך, וזאת כדי למנוע שימוש חוזר ב-Random, שיקל על מתקפות. למעשה, פרט זה לא נאכף והן השרת והן הלקוח שולחים זמנים אקראיים (השרת לא באמת נמצא ב-1992, והלקוח לא באמת נמצא ב-2065...).

ארבעת החלקים שנסקרו עוברים דרך PRF - אותה פונקציה כאילו רנדומלית שסקרנו כשעברנו על רשומות ה-Finished - והתוצאה היא ה-Master Secret.

ה-Master Secret עצמו עדיין אינו סוף הדרך. הוא מתחבר למחרוזת "key expansion" ושוב ל-Random של השרת והלקוח, התוצאה עוברת דרך PRF נוסף, ואז נחתכת לארבעה חלקים.

ההסבר על ארבעת החלקים נתון בפתיחת הפרק, כשסקרנו את ה-Master Secret והמפתחות הנגזרים ממנו.

כעת כשאנו מכירים את ה-Random, אפשר לחזור אל קובץ המפתחות SSLKEYLOGFILE. בקובץ שמורים מפתחות רבים, ו-Wireshark צריך למצוא את ה-Master Secret של סוקט ספציפי. בקובץ המפתחות כל מפתח כתוב ליד ה-Client Random.

שדה ה-SNI

פיתחו את ה-Client Hello ושימו לב לשדה ה-**Server Name Indication**, או בקיצור SNI.

```
▾ Extension: server_name (len=17) name=cyber.org.il
    Type: server_name (0)
    Length: 17
  ▾ Server Name Indication extension
      Server Name list length: 15
      Server Name Type: host_name (0)
      Server Name length: 12
      Server Name: cyber.org.il
```

זהו שדה שמעניין מאד להבין אותו ואת המשמעות שלו. מדוע בעצם יש צורך בשדה הזה? למה הלקוח צריך לציין את שם הדומיין שאליו הוא פונה?

התשובה היא שמרבית הדומיינים בעולם נמצאים על שרתי אחסון משותפים. שרתי אחסון יכולים להכיל דומיינים שונים. חישבו לדוגמה שהשרת המאחסן את cyber.org.il מאחסן גם example.com. מנקודת מבטו של הלקוח, הוא ביצע שאילתת DNS ומצא את כתובת ה-IP של השרת של cyber.orgl.il. מנקודת מבטו של שרת האחסון, כל הפקטות שהלקוח שלח עד כה לשרת היו רק הקמת קישור TCP עם הפורט המתאים, אותו הפורט 443 משרת את כל הדומיינים שהוא מאחסן. הלקוח עדיין לא מסר לשרת האחסון עם איזה דומיין הוא רוצה לתקשר. אם כך, איך שרת האחסון יידע איזה סרטיפיקט להעביר ללקוח? את של cyber או את של example?

נבחן את הדברים בהיבט פרטיות וצנזורה. שם הדומיין שהלקוח פונה אליו עובר בצורה גלויה לחלוטין. כל גורם בדרך יכול לדעת למי הלקוח גולש, ועל סמך השדה הזה ניתן גם לצנזר את הדומיין. ISP יכול לדוגמה לקבל הנחיה שלא להעביר Client Hello שמיועדים ל-cyber.org.il, מה שיחסום את הגישה לדומיין. מה שהלקוח היה רוצה לעשות, הוא שהמידע על איזה דומיין הוא ניגש אליו לא יהיה גלוי. הלקוח היה רוצה להקים

קישור TLS עם שרת האחסון, לסיים איתו Handshake מוצפן, ואז להעביר את המידע על הדומיין המבוקש בצורה מוצפנת. הבעיה היא שבתהליך הקמת הקישור, השרת חייב לשלוח סרטיפיקט, והסרטיפיקט משוייך לדומיין ספציפי.

זוהי אחת הבעיות הפתוחות כרגע ב-TLS. ניתן לעקוף אותה באמצעות שימוש ב-VPN- שדה ה-SNI הגלוי יהיה של שירות ה-VPN ולא של הדומיין הסופי שניגשים אליו.

פתרון עתידי לבעיה הוא ECH, קיצור של Encrypted Client Hello. הרעיון באופן כללי הוא לשנות את פרוטוקול DNS, כך שיכלול לא רק את ה-IP אלא גם את ה-Public Key של **השרת המאחסן**. הלקוח ישתמש בו כדי להצפין את ה-Client Hello כולו, או אפילו רק את שדה ה-SNI. השרת המאחסן יוכל לפתוח את ה-SNI ולראות לדוגמה שהלקוח ביקש את cyber.org.il. משם ה-Handshake ימשיך כרגיל.

RTT

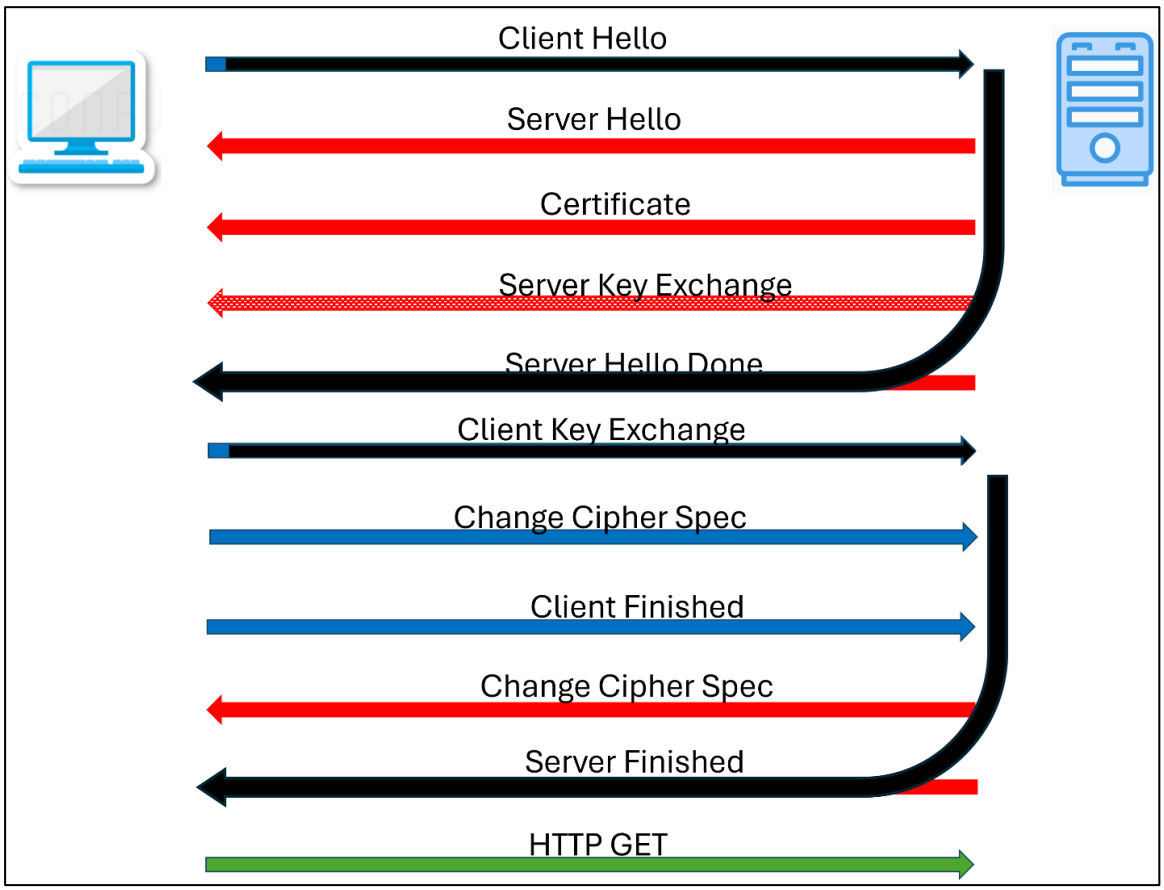
לפני שנעבור לנושא הבא, תהליך Handshake מקוצר, עלינו להבין נושא מרכזי בתהליך ה-Handshake והוא Round Trip Time, או בקיצור RTT.

RTT הוא פרק הזמן שלוקח מרגע שצד אחד לתקשורת שולח פקטה ועד שהתשובה עליה מתקבלת. דוגמה פשוטה לכך היא פינג, פרוטוקול ICMP שלמדנו. הלקוח שולח פינג לשרת ומודד כמה זמן לוקח עד שהתקבלה תשובה.

כיוון שפרק הזמן בין בקשה לתשובה משתנה בין קישור לקישור, תלוי במרחק שבין השרת והלקוח וכמות הרכיבים ביניהם, מודדים מהירות של פרוטוקולים באמצעות יחידות של RTT. כלומר, לא אין לנו יכולת לנבא מראש כמה זמן ייקח לסיים לדוגמה תהליך Handshake של TLS, אבל אנחנו כן יכולים להגיד כמה RTT יש בין תחילת ה-Handshake לסופו. מה דעתכם, כמה ישנם?

לטובת המספור, חשוב להבין שמספר רשומות שנשלחות בפקטה אחת לא משנות את ה-RTT (כן, אנחנו מניחים שרוב הזמן הוא בהעברת הפקטות, לכן פקטה ארוכה תיקח בקירוב אותו זמן כמו פקטה קצרה). יתרה מכך, מספר פקטות שנשלחות את אחרי השניה, בלי שהגורם השולח צריך להמתין לתשובה, גם לא יעלו את ה-RTT. אם אין המתנה לתשובה מהצד השני, אפשר להתייחס לכל כמות של פקטות בתור פקטה אחת ארוכה.

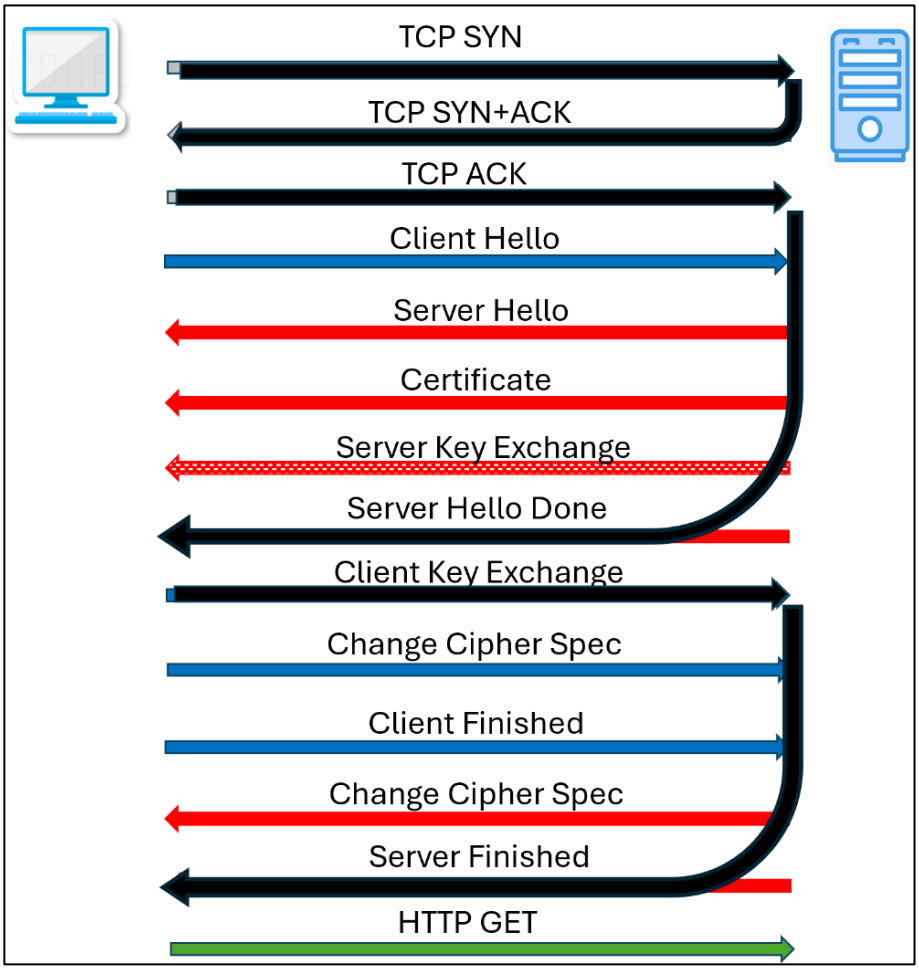
אם כך נספור כמה RTT-ים יש מרגע שהלקוח שולח את הפקטה הראשונה ב-TLS, עד שהלקוח יכול לשלוח את הפקטה הראשונה של פרוטוקול שכבת האפליקציה, לדוגמה בקשת HTTP GET:



הלקוח שולח Client Hello ונאלץ להמתין עד ה-Server Hello Done. זהו RTT אחד. הלקוח שולח Client Key Exchange ורשומות נוספות, ונאלץ להמתין עד ה-Server Finished. זהו RTT נוסף. כלומר לאחר שני RTT הלקוח יכול להתחיל בשליחת HTTP GET. לכן ה-TLS Handshake של גרסה 1.2 נחשב "2 RTT".

חשוב להדגיש: הספירה של ה-RTT-ים של TLS לוקחת בחשבון רק את הרשומות של TLS. כאשר TLS עובר מעל TCP, מתרחש לפניו תהליך ה-3Way Handshake של TCP.

תהליך זה מוסיף כמובן RTT נוסף:



תהליך ה-3Way Handshake מוסיף RTT יחיד, למרות שהוא שלוש פקטות, מכיוון שהלקוח לא צריך לעשות הפסקה בין השליחה של הפקטה האחרונה, ה-ACK, לבין ה-Client Hello.

אם כך, מדוע מודדים בנפרד את ה-RTT של TLS ושל TCP? האם לא היה מובן יותר לחבר אותם יחד ולומר "תהליך ה-Handshake של TLS בגרסה 1.2 הוא 3-RTT, מתחילת יצירת הקשר ועד שעובר מידע של שכבת האפליקציה"?

ובכן יש סיבה טובה מדוע מפרידים את ה-RTT של TLS מה-RTT של TCP. כאשר נלמד על פרוטוקול QUIC נבין זאת. בינתיים, נסקור שינויים ושיפורים ב-RTT של TLS, גם בגרסה 1.2 וגם בהמשך בגרסה 1.3.

Session Resumption

כפי שראינו, תהליך ה-Handshake הוא 2-RTT. לעיתים אין ברירה אלא לבצע אותו, אבל מה אם כרגע סיימנו גלישה לאתר כלשהו, סגרנו את הדפדפן ואז החלטנו להיכנס שנית?

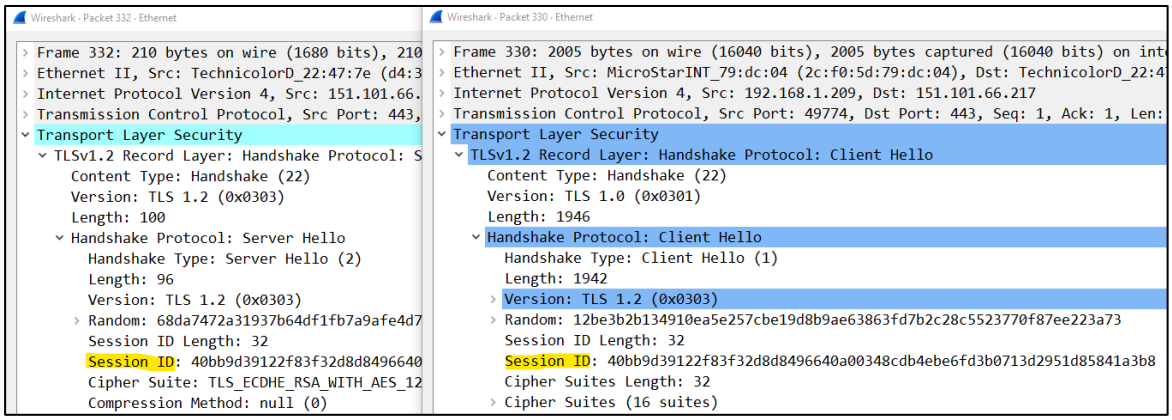
פיתחו את ה-Client Hello ואת ה-Server Hello ותמצאו שם שדה Session ID. שדה ה-Session ID מסייע לשרת וללקוח להקים סוקט מאובטח מהיר תוך דילוג על חלק משלבי ה-Handshake.

בתקשורת הראשונה אי פעם עם השרת, הלקוח יכניס ערך 0 ל-Session ID. השרת יענה לו עם Session ID כלשהו, ייחודי ללקוח הנ"ל.

בפעם הבאה שהלקוח יפנה לשרת, הלקוח ישלח לשרת את ה-Session ID שהשרת סיפק לו. כעת ישנן שתי אפשרויות. השרת יכול להגיד "אני לא זוכר אותך, או שה- Session ID שלך ישן מדי. בוא נתחיל Handshake מחדש". במקרה זה השרת ישלח Session ID שונה מאשר מה שהלקוח שלח לו, ויבוצע Handshake מלא. לחילופין, השרת יכול להגיד "היי! ברוך הבא שוב. בוא נקצר עניינים" ולהשיב ללקוח עם אותו Session ID שהלקוח שלח לו ב-Client Hello.

בהסנפה שלנו מול www.cyber.org.il הלקוח הציע לשרת Session ID קודם ביניהם, אך השרת בחר להתעלם מהאפשרות להאיץ את התהליך. אם תרצו לראות תהליך מקוצר, בצעו גלישה לאתר כרצונכם ועשו refresh לדפדפן (F5).

אם תפתחו את ה-Session ID של ה-Client Hello ושל תשובת השרת, ה-Server Hello, תוכלו לוודא שיש להן את אותו ערך (0x40bb9d...):

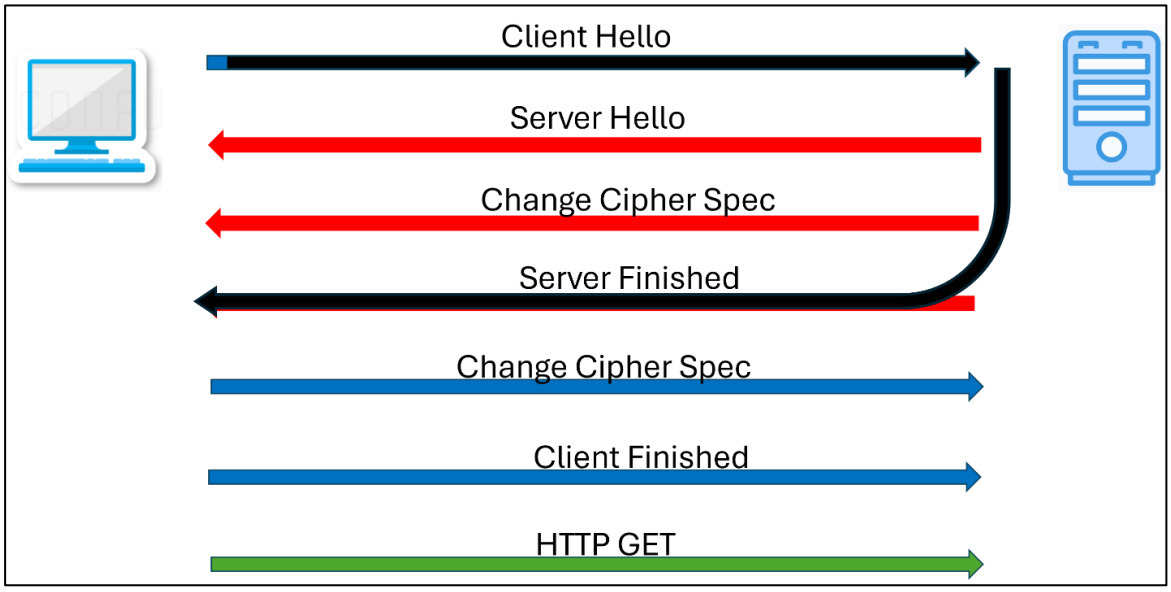


בעקבות תשובת השרת שענה עם אותו ID, המשך התהליך היה שונה וקצר יותר:

No.	Time	Source	Destinat	Protocol	Length	Info
330	4.316467	192.1…	151.1…	TLSv1.2	2005	Client Hello (SNI=www.themarker.com)
332	4.374574	151.1…	192.1…	TLSv1.2	210	Server Hello, Change Cipher Spec, Finished
333	4.374848	192.1…	151.1…	TLSv1.2	105	Change Cipher Spec, Finished

רואים שכמות ה-Records שהוחלפו היא קטנה יותר (לדוגמה, השרת לא העביר סרטיפיקט ללקוח).

נבדוק מה קרה ל-RTT:



הלקוח שלח Client Hello וחיכה עד ה-Server Finished. RTT אחד.

הלקוח שלח רשומות נוספות, אך לא חיכה לתשובת השרת וכבר יכל לשלוח את ה-HTTP GET. במילים אחרות, מרגע שהחל את התהליך ועד שיכל לשלוח HTTP GET, הלקוח המתין RTT אחד. חסכנו RTT.

Session Ticket

הרעיון של Session Resumption הוא מצוין, אבל המימוש על ידי Session ID הוא בזבזני מבחינת השרת. שימוש ב-Session ID מאלץ את השרת להחזיק בזיכרון את כל ה-Session ID של לקוחות שסיימו התחברות אליו ואת כל המידע שצריך בשביל יצירת המפתחות מחדש: ה-Master Secret ושני ה-Random-ים. בלעדיהם, השרת ייאלץ לבצע תאום חדש של סוד משותף.

העניין הוא ששרתים לא אוהבים להחזיק אצלם מידע של לקוחות. זה מעמיס על השרת וגם מאפשר ניצול לרעה - לדוגמה גורם זדוני עלול להעמיס את השרת בכמות גדולה של Session ID ומפתחות לשמור. שרתים מעדיפים שהלקוח יעמיס על עצמו את שמירת המידע הנדרש.

פיתחו את ה-Client Hello וחפשו שם את ה-Session Ticket:

```
Extension: session_ticket (len=192)
    Type: session_ticket (35)
    Length: 192
    Session Ticket […]: d5fce2961f9af00c2ecbb4f723fc9d050aebf
```

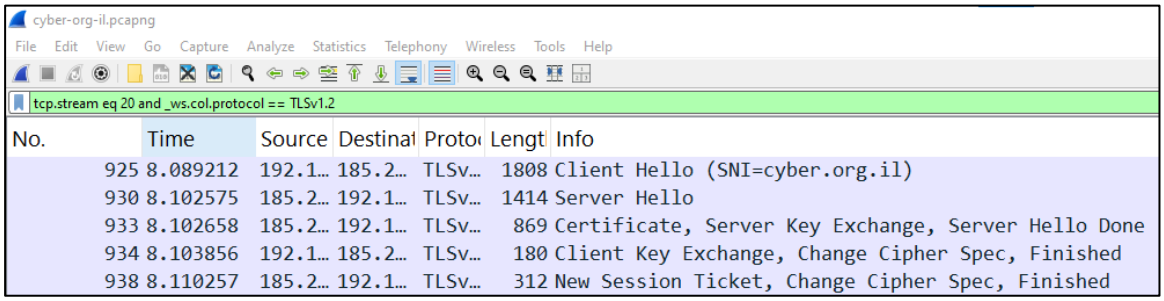
ה- Session Ticket כולל:

- פרק זמן שבו הוא בתוקף
- גרסת ה-TLS שתואמה בין הצדדים
- ה-Cipher Suite שתואם
- ה-Master Secret שתואם
- ה-Random-ים של השרת והלקוח

רגע, אם שולחים את כל המידע הזה בגלוי, כל אחד יכול לפענח את התקשורת בין השרת והלקוח. מה נעשה? הנה החלק הנחמד - כל המידע מוצפן באמצעות המפתח הציבורי של השרת, כך שרק השרת יכול לפענח אותו.

מהיכן מגיע ה-Session Ticket?

כעת נחבר את החלק האחרון בפאזל של תהליך ה-TLS 1.2 Handshake. חיזרו אל קובץ ההסנפה של cyber.org.il. דילגנו על רשומה אחת ששלח השרת ללקוח - New Session Ticket.



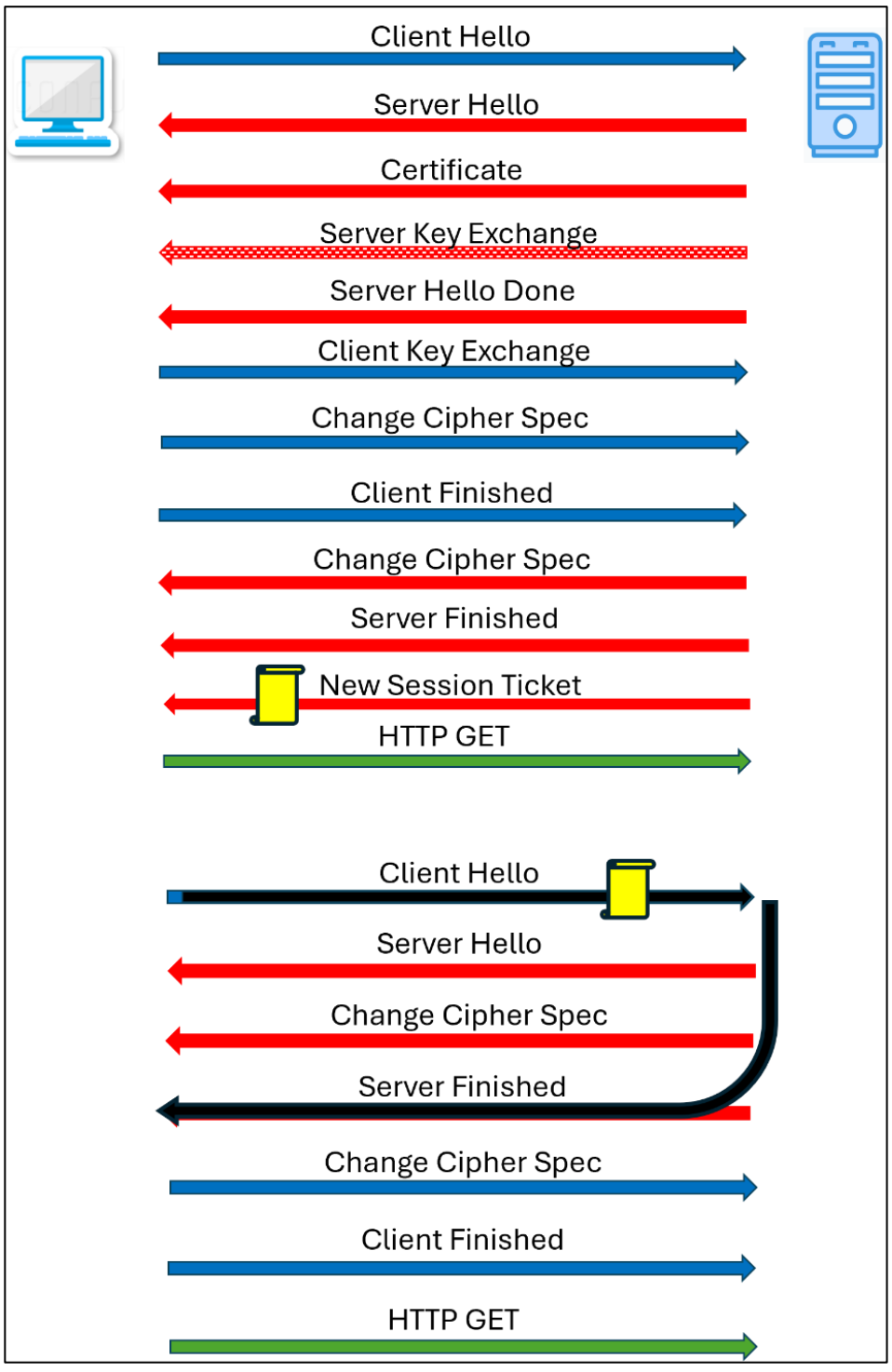
ברשומה האחרונה לפני שהשרת מתחיל להצפין, הוא שולח ללקוח את הפרטים הנדרשים, למקרה שהלקוח ירצה לחדש תקשורת מולו בזמן הקרוב:

```
▾ Handshake Protocol: New Session Ticket
    Handshake Type: New Session Ticket (4)
    Length: 198
  ▾ TLS Session Ticket
      Session Ticket Lifetime Hint: 300 seconds (5 minutes)
      Session Ticket Length: 192
      Session Ticket […]: 50c2c995532766015f74b6a8f55ba91db2f
```

כעת הפרטים שמורים אצל הלקוח והשרת לא נדרש לשמור אצלו דבר לאחר סיום ההתקשרות.

תהליך ה-Handshake נראה דומה מאד לתהליך שמתרחש ב-Session Resumption, פרט לכך שהלקוח משתמש ב-Extension בשם Session Ticket. במידה והשרת מקבל את בקשת הלקוח לעשות את קיצור הדרך, השרת יוסיף ל-Server Hello גם Extension בשם Session Ticket, שמאשר את הבקשה.

הנה כך נראה התהליך, כולל שימוש ב-Session Ticket. תחילה, תהליך Handshake רגיל של 2-RTT. השרת מעביר ללקוח New Session Ticket. בהתקשרות הבאה, הלקוח משתמש בו ומתקיים Handshake 1-RTT:



[RTT TLS 1.2 Session Ticket-1]

תרגיל פענוח הסנפת TLS

הורידו את הקובץ הבא:

https://data.cyber.org.il/networks/TLS12files.rar

תחת הקובץ נמצאים קובץ הסנפה וקובץ מפתחות.

ענו על השאלות הבאות:

1. כמה פקטות מכילות Client Hello? השתמשו בפילטר כדי להשאיר רק Client Hello.
2. התמקדו בפקטה הראשונה שמכילה Client Hello. לאיזה דומיין מבוצעת הגלישה?
3. צרו פילטר שמפלטר רק Client Hello אל הדומיין הספציפי שמצאתם בסעיף הקודם
4. פלטרו את כל הרשומות בתהליך ה-Handshake שהוא ההמשך של ה-Client Hello מהסעיף הקודם. איך ניתן, בהתבסס על שמות וסדר הרשומות שנשלחות בלבד ובלי לבדוק את ה-Server Hello, לדעת אם ה-Handshake הוא DH או RSA?
5. מהו ה-Cipher Suite שנבחר על ידי השרת? איזה אלגוריתם נבחר עבור:
    a. Authentication
    b. Symmetric Encryption
    c. Hashing
    d. Key Exchange
6. כמה סרטיפיקטים נשלחו על ידי השרת? מה ה-Common Names של הבעלים של הסרטיפיקטים הללו?
7. האם הסרטיפיקט ששלח השרת תקף לכל שרת תחת הדומיין, או לשרת ספציפי?
8. בין אילו תאריכים הסרטיפיקט בתוקף?
9. מיהו ה-Root CA? האם הסרטיפיקט שלו נשלח?
10. הוסיפו את קובץ המפתחות. איזה משאב הלקוח מבקש בבקשת ה-GET הראשונה שלו? מהו ה-Status Code המוחזר מהשרת?
11. מהו שם המשתמש והסיסמה שנשלחו מהלקוח אל השרת? לאחר שביצעתם Follow TCP Stream חפשו את המחרוזות Username, Password.

סיכום

התחלנו את הפרק בכך שהתקשורת בין השרת והלקוח עוברת מעל רשומות. סקרנו את המבנה של רשומה ואת סוגי הרשומות הקיימות.

מכאן התמקדנו ברשומות של Handshake. הבנו כי הדבר הראשון שהשרת והלקוח צריכים לתאם הוא ה-Cipher Suite. בחירה זו כוללת את השיטה הנבחרת להחלפת מפתחות סימטריים. ראינו כי ל-DH ול-RSA יש שתי גרסאות שונות במקצת של Handshake.

סקרנו כיצד לאחר בחירת שיטת החלפת המפתחות הצדדים יוצרים סוד משותף, ה-Pre Master Secret, ממנו נגזר ה-Master Secret, ממנו נגזרים ארבעת המפתחות שנעשה בהם שימוש בסוקט המאובטח: מפתח הצפנה סימטרי לשרת, מפתח הצפנה סימטרי ללקוח, מפתח HMAC לשרת ומפתח HMAC ללקוח.

ראינו כיצד באמצעות ה-Finished השרת והלקוח מוודאים שאף גורם לא התערב ביניהם ושיחק ברשומות.

לבסוף ניתחנו את כמות ה-RTT בתהליך ה-Handshake של גרסה 1.2. ראינו כי נדרש 2-RTT וראינו כי באמצעות Session Ticket אפשר לקצר את התהליך ל-RTT בודד.

על הדרך ביצענו יצירה של קובץ מפתחות SSLKEYLOGFILE וייבאנו אותו לתוך Wireshark. בפרק הבא נקפוץ עשור קדימה, מ-TLS 1.2 של שנת 2008 אל TLS 1.3 של 2018.