על תעודות דיגיטליות - איך לסמוך על הסינים באינטרנט

מאת יהונתן אלקבס

הקדמה

יצא לכם פעם לחשוב על זה שאנשים אשכרה שמים את פרטי האשראי שלהם (ו/או כל מידע רגיש אחר) באינטרנט כאילו זה סבבה? איך מישהו יכול לוודא שהוא באמת שולח את המידע לאתר שהוא סומך עליו ולא לחיקוי ו/או העתק זדוני שלו? למה אנחנו כל כך משוכנעים שמישהו לא יכול להאזין לתקשורת שלנו ולחלץ ממנה מה שהוא רוצה?

ניקח דוגמא מהחיים - אני משתמש ב-PayPal, למעשה אני ממש חובב PayPal, איך אני יכול לדעת ב-100% שהפרטים שלי באמת הולכים לחברה שמנהלת אותו - PayPal Holdings ולא לאיזה ישות או אתר שוואלה נראה כמו PayPal אבל הוא לא? מאיפה לי הביטחון ש-AliExpress (או תוקף אמיתי) לא מנתר אחרי התעבורה שלי (או של PayPal) ויכול לחלץ את פרטי האשראי שלי משם? התשובה הקצרה - כי האתר של PayPal **מוכיח** לדפדפן שלי שהוא באמת מייצג את PayPal Holdings והתעבורה בנינו מוצפנת בטירוףףף.

התשובה הארוכה היא: **HTTPS**.

התשובה הארוכה יותר הוא: **SSL\TLS**

והתשובה המלאה היא: **תעודות דיגיטליות ומפתחות**.

מקווה שסקרנתי אתכם קצת. בדוגרי, מטרת המסך היא פשוטה - להביא את כל מי שקורא אותו **מ-0 ל-100** בכל הקשור לנושא של תעודות (ותעודות X.509 בפרט) ובכללי מאיך שעובדת תקשורת מבוססת הצפנה באינטרנט (HTTPS). אנחנו הולכים לעבור ולנתח הרבה נושאים שונים, החל מלפתוח תעודות ולעבור על התוכן שלהן בפורמט ASN.1, לנתח פאקטות TLS ב-Wireshark, לדבר על PKI, להכין בעצמנו תעודות X.509, לחתום עליהן ולהראות את השימוש הפרקטי בהן, לדבר לא מעט על חולשות בארכיטקטורה שנוכחת כיום וגם להמליץ איך לאבטח תעודות ו-TLS בצורה חכמה ונכונה.

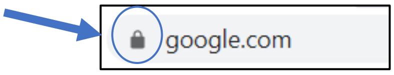
חשוב לי מאוד לציין כי הנחת היסוד היא שכל מי שקורא את השורות הללו כעת, לא מגיע עם רקע מקדים בכלל, **ממש כלום**. אני מאמין שכאשר רוצים להסביר למישהו משהו בצורה טובה חייבים להתחיל מההתחלה, גם אם זה אומר לחזור טיפה על משהו שהוא כנראה יודע. כפי שאוהבים לשאול באינטרנט:

Explain to me like I’m 5 years old

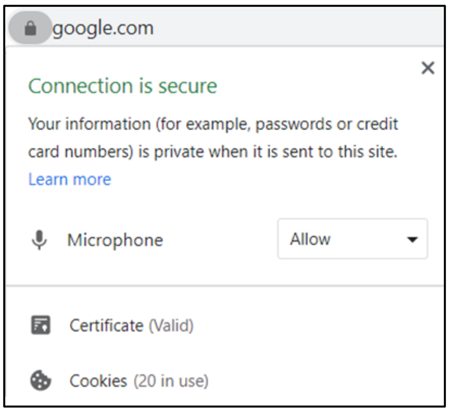
כפי שבוודאי יצא לכם להבין מההקדמה (ומאורך המסמך), אנחנו נעבור על המון (אבל המוווון) נושאים. חלקם אתם בוודאי מכירים וחלקם אולי פחות, תרגישו חופשי לרפרף על מה שאתם בטוחים בו. בכל מקרה, הייתי ממליץ כבר עכשיו להרתיח מים למשקה האהוב עליכם (זה בסדר גם אם זה תה, אני לא שופט) ולהתפנק במנת סוכר טובה.

תעודות רבותיי תעודות

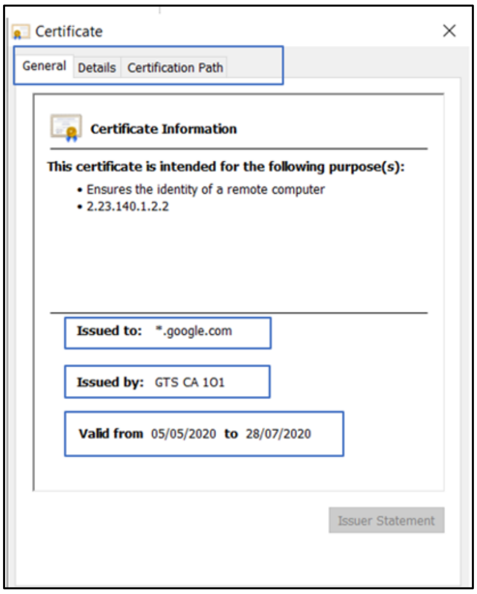
טוב, תעודות. מה זה תעודה בכלל? הדרך הכי פשוטה היא פשוט להראות אחת. על מנת לבחון תעודה נבחר בדפדפן Google Chrome (כל דפדפן תומך בפונקציונאליות דומה) וניכנס לאתר אינטרנט שעושה שימוש ב-HTTPS, google.com לדוגמא. נלחץ על האייקון של המנעול בשורת ה-URL:



בחלון שיפתח נבקש להתבונן בתעודה:



עם לחיצה על השורה היא תפתח:



נוכל לראות מאפיינים כלליים על התעודה כגון למי היא שייכת, הישות שהנפיקה אותה והתאריכים בהם היא בתוקף. עם התקדמות המסמך נעבור וננתח מאפיינים נוספים.

עכשיו תורכם - כנסו לאתר האהוב עליכם (אפשר גם ב-url של הגיליון) ותנסו לראות איזו תעודה הוא מציג!

אם כך - מהי תעודה?

במשפט אחד, תעודה משמשת כ-container למפתח הציבורי של בעל התעודה ומכילה חתימה דיגיטלית המספקת ערבות כי אכן מחזיק התעודה הוא מי שהוא טוען שהוא. בנוסף, קיימים מספר שדות (הרחבות) המגדירים כיצד יש להעביר את התעודה, איסורים על שימוש במפתח שנמצא בה ופרטים רבים נוספים.

נשמע מורכב? אנחנו הולכים לפרק את המשפט הזה ולהבין אותו בצורה טובה מאוד בתת הפרק הבא.

צוללים קדימה: קריפטוגרפיה והצפנות

קריפטוגרפיה הוא תחום העוסק בהסתרה. למשל, הסתרת תקשורת בין 2 ישויות מגורם צד שלישי. אם ה-router שלי שולח פרטים אישיים שלי (כמו מס' כרטיס אשראי או תמונות מביכות מכיתה ד') ל-Router של PayPal, לא הייתי רוצה שמישהו יוכל לשבת על התווך התקשורתי שבניהם ולהאזין לתעבורה ולחלץ אותם.

בסופו של דבר, כל "סוד" במרחב מדעי המחשב **מתחיל ונגמר במפתחות**.

קריפטוגרפיה של מפתח ציבורי, על רגל אחת, מסתמכת על מפתחות להצפנת ופענוח של מידע. המפתחות קשורים באופן מתמטי, כלומר, המידע מוצפן באמצעות אחד מהמפתחות וניתן לפיענוח רק באמצעות האחר. אם נעשה שימוש במפתח יחיד גם לקידוד וגם לפיענוח מדובר **בהצפנה סימטרית** ואם מדובר על צמד מפתחות (ציבורי ופרטי) נאמר כי מדובר **בהצפנה אסימטרית** - בעוד שהמפתח הפרטי נשמר בסוד, המפתח הציבורי מוטבע בדרך כלל בתעודה דיגיטלית, והתעודה מתפרסמת במסד נתונים אליו כל המשתמשים המורשים יכולים לגשת.

במשפט אחד - *הצפנת מפתח ציבורי היא שיטה בקריפטוגרפיה המאפשרת לגופים מרוחקים לתקשר באופן מאובטח ברשת בעלת תווך לא מאובטח*. כמו התווך בין ה-Router שלי ל-Router של PayPal.

הבעיה העיקרית בקריפטוגרפיה של מפתח ציבורי היא העובדה **שאין קישור בין המפתח הציבורי ובעליו**. קרי, תוקף פוטנציאלי יכול להתחזות לאדם אחר תוך שימוש במפתח הציבורי שלו. כיצד נוכל לאמת מי נמצא בצד השני? האם המפתח שקיבלנו באמת מהשולח שאנחנו חושבים? האם המפתח עדיין בתוקף?

שאלות רבות צצות במהרה ומתברר שקריפטוגרפיה של מפתח ציבורי אכן מעניקה מענה לבעיית אבטחת המפתח הציבורי עצמו, אך לא את הבעיות של *רכישת מפתח ציבורי*, *הכרה*, *ביטול*, *הפצה*, *חלוקה*

מחודשת, *אימות*, והכי חשוב, *ייחוס המפתח למערכת מזהה - ליישות בעולם האמיתי*. כלומר, לא ניתן לאמת את מקור התקשורת ואת שלמותה עם מפתחות בלבד.

שורה תחתונה - **תקשורת יכולה להיות פרטית אך לא מאומתת.**

עולה הצורך בגוף ייעודי שיהיה מסוגל לגשר על הפערים האלו. גוף שידע לסווג מפתחות לבעליהם החוקיים ושממשקים רבים יוכלו לסמוך עליו. צריך להיכנס משתנה חדש למשוואת האבטחה - **אמון**.

נתינת אמון בגורם צד שלישי היא דרישה בסיסית לכל יישום בהיקף נרחב של מוצר אבטחת רשת המבוסס על הצפנה באמצעות מפתחות ציבוריים. ישנם מספר פתרונות למימוש האמון, אך הפתרון הנפוץ ביותר (עקב הנחות העבודה הבסיסיות שהוא דורש) הוא שימוש בתעודות, *סרטיפיקטים* בלעז. במרחב המופלא של האינטרנט תראו לא מעט אנשים משתמשים בקיצור cert בדיבור על תעודות.

Cert אחי Cert!

הנחת העבודה היא שבמערכת ישנה רשות CA (Certificate Authority) **שכולם סומכים עליה** (ומכירים את המפתח הפומבי שלה) ועבודתה להנפיק תעודות למשתמשים לאחר שהיא אכן ביררה שמי שפנה אליה אכן מחזיק בזהות שהוא מתיימר להחזיק.

אנאלוגיה שכיחה לנושא היא נשיאת דרכון במעבר בין מדינה למדינה. דרכון הוא תעודה אשר חתומה על ידי רשות חוק בדרג גבוה (הממשלה) המאשרת כי מחזיק התעודה (אזרח) הוא אכן מי שהוא טוען שהוא. כל מדינה אחרת שסומכת על כושר אכיפת החוק ועל סמכות הממשלה של המדינה שחתומה על הדרכון **תאשר גם היא** את זהותו של האזרח.

תעודה היא בסך הכל מחרוזת שמכילה מס' שדות אודות מחזיק התעודה (כגון שמו), תאריך התפוגה של התעודה, שם ה-CA שהנפיק אותה ו**המפתח הפומבי של הלקוח.** (בהמשך המסמך נתעמק היטב בכל שדות התעודה).

נדגים שימוש פרקטי בתעודות - כאשר משתמש A רוצה לשלוח למשתמש B את המפתח הפומבי שלו, הוא שולח אליו את הסרטיפיקט שלו. מכיוון שהתעודה מכילה את המפתח הפומבי של A בתוכה הרי ש-B קיבל גם את המפתח הפומבי של A.

אם B סומך על ה-CA שהנפיק את התעודה, הוא מאמת את החתימה של ה-CA על הסרטיפיקט תוך שימוש במפתח הפומבי של ה-CA (שכאמור ידוע לכולם) וכך מוודא שהמפתח הפומבי אכן שייך ל-A.

חשוב לציין שה-CA לא יודע לפענח או לחתום באמצעות המפתחות שאותם הוא מאשר! כלומר, הוא לא מאמת את **תקינות** המידע שלקוח מעביר אלא הוא רק מאמת את זהותו של הלקוח.

אין צורך בפיתולים מחשבתיים בשביל להבין שקיימת בעיה מעשית בהנחת העבודה שמצדיקה את קיומו של הCA - בהינתן רשת מספיק גדולה, קיים סיכוי סביר ששני ישויות קצה A,B לא מכירים CA אחד ששניהם **מכירים וסומכים** עליו. אם כן מה ניתן לעשות?

PKI וחברים

PKI (**Public Key Infrastructure**) היא תשתית של חומרה, תוכנה, אנשים, מדיניות ותהליכים הנדרשים כדי להנפיק, לנהל, להעביר, להשתמש, לאחסן ולגנוז תעודות דיגיטליות ובכך מאפשרים תשתית תומכת למפתחות ציבוריים.

למה אני מספר לכם על זה?

כשמדברים על PKI בעצם מדברים על ההסכם שמקשר **מפתחות ציבוריים עם הישויות הפיזיות** שאליהם הם שייכים באמצעות CA. הקשר מיוסד בתהליך של רישום וניפוק בהתבסס על אמון בין הצדדים (למשל על ידי הצגת מסמכים משפטיים המאשרים את זהות החברה). הגוף שאחראי על בדיקות האמון הללו נקרא RA (Registration Authority). על ה-RA לוודא שהמפתח הציבורי אכן שייך לאותו גוף יחיד אשר נרשם, באופן שלא יאפשר לאותו גוף להתכחש לכך בעתיד. כדי לדעת אם תעודה כלשהי בתוקף פונים ל-VA (Authority Validation) אשר מאגד את תיקוף התעודות. על מנת לפשט את קריאת המסמך, מעתה נתייחס ל-RA ו-VA כאילו מדובר ב-CA עצמו אבל יש לזכור שהמצב שונה.

מרבית ה-PKI-ים מכילים מספר CA-ים, כאשר גם משתמשים יכולים לשמש כגורמים מאשרים (נרחיב בהמשך על הנושא) וכל CA יכול להנפיק תעודה ל-CA אחר או למשתמש קצה. משתמשים שלא משמשים כגורמים מאשרים נקראים EE (**End Entities**).

פורמטים לאחסון תעודות

ישנם מספר פורמטים שונים לתעודות כתלות באופן ישיר בו המידע של התעודה מאוחסן. סוגי הפורמטים השכיחים ביותר:

- **DER/CER** - Distinguished Encoding Rules (DER) ו-Canonical Encoding Rules (CER). קבצים בפורמט בינארי לאחסון מידע בתעודות. עם סיומות הקבצים: **.cer / .der** בהתאמה. מכיוון שקבצי DER מיוצגים בינארית יהיה מאתגר לנתב אותם במערכות (כגון שרתי מייל) שתומכות רק ב-ASCII.

- **PEM** - Privacy-enhanced Electronic Mail הוא קובץ בפורמט **ASCII** שיכול להכיל הרחבה של סיומת **.pem**, **.crt**, **.cer** ו-**.key**. קבצי PEM מאוד נפוצים והם מתחילים ב:

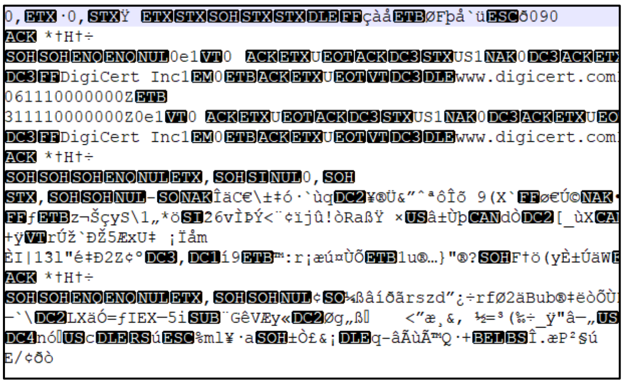
```
-----BEGIN CERTIFICATE-----
```

ומסתיימים ב:

```
-----END CERTIFICATE-----
```

- **PFX/P12** - Personal Information Exchange, ידוע גם כ- PKCS#12 הוא קובץ בינארי בסביבת עבודה של Microsoft ליבוא ויצוא של תעודות. הסיומות של קבצי PFX הן **.pfx** ו-**.p12**.

נציין כי מכיוון שקובץ PEM מיוצג על ידי ASCII (בקידוד Base64) נוכל להסתכל על 'תכולתו' באמצעות תוכנת טקסט כגון ++Notepad או כתבן ומכיוון שקבצי DER\CER\PFX הם קבצים בינאריים נקבל "ג'יבריש" כשננסה לקרוא אותם (...and Hello Captain Obvious).



למה לי להאמין לך? מסלולי אמון ב-PKI

הדרכים בהם CA מאורגנים, מי חתם למי, כיצד נראה מסלול אמון ואיך מוצאים אותו הינם אבני יסוד בהגדרת PKI. ארכיטקטורת המבנים הנפוצים:

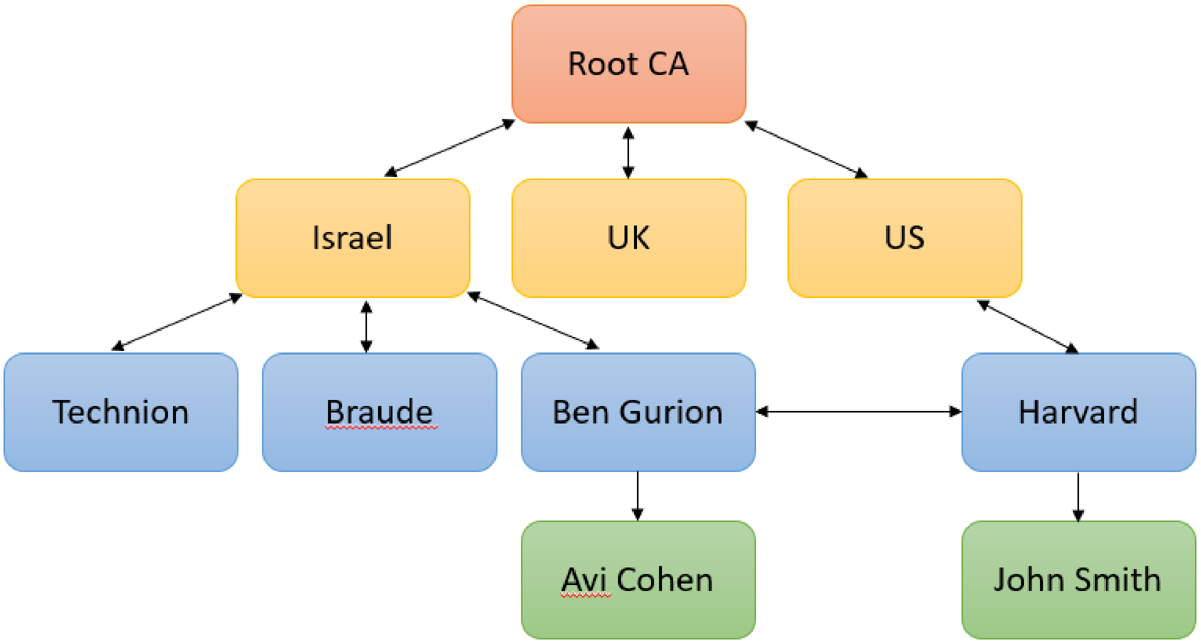
- **PKI היררכי**: ה-CA-ים מאורגנים בעץ כאשר כל צומת פנימית בעץ חותמת לילדיה ולאביה, והעלים הם EE.
- **PKI קליק** כל CA חותם לכל הCAים האחרים ובכך מבטיחים שאורך מסלול האימות יכיל תמיד רק שני CA.

ישנם מבנים נוספים שהם הכלאה בין 2 הגישות - סוג של עץ כללי אבל עם חתימות נוספות שמהוות קיצורי דרך. מונח זה נקרא cross certification ונקדיש לו את תשומת הלב בהמשך המסמך.

ישנם PKI-ים <u>ללא מבנה מסודר</u> של CA. למשל, ב-**PGP** (Pretty Good Privacy) אין מבנה בכלל. כל משתמש משמש גם כ-CA ויכול לחתום על תעודות של משתמשים אחרים. במקרים אלו על כל משתמש לאמץ מדיניות לפיה יקבל או ידחה תעודות. מבנה זה נקרא **Web of trust**. המבנה נוח מכיוון שבין כל זוג

משתמשים יש מסלול אימות קצר ולכן לכאורה המערכת יותר גמישה. מצד שני, מכל משתמש מצופה עבודה רבה בהחלטה על איזה מסלולים אמונים עליו.

דוגמא לארגון CA-ים:



בסיכומו של יום, הבעיות שמבנה ה-PKI צריך לתת עליהן את הדעת הן:

- Scalability - התאמה למקרים שונים ולהתפתחויות שונות של הרשת (כמות משתמשים גובהה, משתמשים רחוקים פיזית וכו')
- בטיחות - עד כמה פריצה לאחד מה-CAים בעלת פוטנציאל הרס בתפקוד הרשת
- רגישות - האם כל EE שווה זכויות או שקיים צורך בהפרדה
- אמון
- פרטיות
- סיבוכיות
- סוגיות חוקיות (ביטוח)
- התהליך שיש לעבור להנפקת תעודה, עדכונה וביטולה.

למרות שהשימוש ב-CA שמנפיק תעודות ומבטיח את הגנה על המפתח הציבורי, קיימת בעיה נוספת - *כל תעודה יכולה להיות בפורמט שונה.* מה אם המכונה שאני עובד איתה לא מסוגלת לקרוא ולעבד תעודות של חברה כזו או אחרת? אי לכך, נקבע כי הפורמט האחיד יהיה בתצורת **תעודה X.509.**

תקן X.509

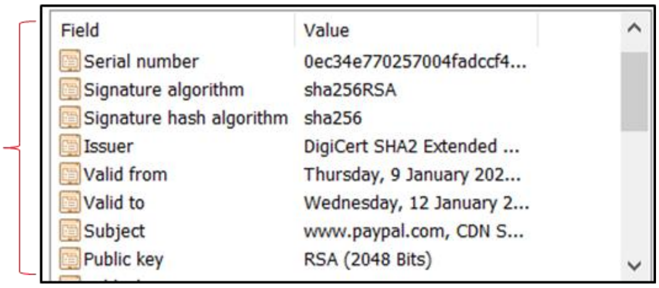
תקן X.509 הוא תקן ITU-T **שמגדיר PKI**. כל מה שקראנו עד כה רלוונטי עבורו וממומש על ידו. התקן מגדיר את דרישות הסף שנדרשות להגדרה איכותית ובטוחה של תעודות. התקן מפרט בין היתר שיטות לתיעוד, שיוך, אישור, פסילה ואימות מפתחות הצפנה אסימטריים.

משהו שחשוב להבין - יש המון אבל המוווון RFC-ים (A Request for Comments - מסמכים שנכתבים בדרך כלל על ידי מהנדסים וחוקרים בתחום מדעי המחשב לשם הפצת תפיסות חדשות ומידע. מחזיק אצבעות שאפעם לא תצטרכו לקרוא\ לפענח אחד כזה) בנושא תעודות.

RFC 5280 מגדיר את הפורמט של תעודות X.509 ואת אופן ביטולן (**RCL**), RFC 4210 מגדיר את פרוטוקול ניהול התעודות (**CMP**), RFC 4158 מגדיר את בניית שרשרת התעודות (**certificate chaining**), ולבסוף, RFC 3739 מגדיר את מה שמכשיר את התעודה (Qualified Certificate בלעז) כדוגמת מבנה **ASN.1** (קיימים עוד RFC אודות הנושא אבל אלו פה הם אלה שחשוב לנו להכיר).

כאמור, קיים חומר רב אודות תעודות X.509, מטרת הפרק הנ"ל לתמצת את העיקר ולספק ידע תיאורטי בסיסי והבנה כללית על אופן הניהול של התעודות.

תעודות X.509 משמשות בפרוטוקולי אינטרנט רבים, כדוגמת TSL/SSL שהוא הבסיס ל-HTTPS המאבטח גלישה באינטרנט. תעודה X.509 מכילה מפתח ציבורי וזהות (שם מארח, ארגון או יחיד), והיא חתומה על ידי CA או על ידי בעל התעודה עצמו (הרחבה בהמשך). כאשר תעודה נחתמת על ידי CA מהימן, או מאומת באמצעים אחרים, מישהו שמחזיק בתעודה הזו יכול לסמוך על המפתח הציבורי שיש בו כדי ליצור תקשורת מאובטחת עם אותו גורם, או לאמת מסמכים שנחתמו באופן דיגיטלי על ידי המפתח הפרטי.



Field	Value
Serial number	0ec34e770257004fadccf4...
Signature algorithm	sha256RSA
Signature hash algorithm	sha256
Issuer	DigiCert SHA2 Extended ...
Valid from	Thursday, 9 January 202...
Valid to	Wednesday, 12 January 2...
Subject	www.paypal.com, CDN S...
Public key	RSA (2048 Bits)

אם הדפדפן שלי יקבל תעודה מ-PayPal ש-DigiCert בכבודו ובעצמו ערב לביטחונה הוא לא יפקפק אפילו למילי-שנייה על אמינותה ויאשר את הטעינה של דף האינטרנט (ה-DOM עצמו) של PayPal.com.

תקן X.509 אמון גם על הגדרת רשימות לביטול תעודות (**CRL**), המהוות אמצעי להפצת מידע על תעודות שסומנו כלא חוקיות על ידי ה-CA, וכן אלגוריתם אימות לתעודות, המאפשר חתימה על תעודות על ידי ישות ביניים (intermediate CA). ישויות ביניים בתורן, חתומות על ידי תעודות אחרות, ובסופו של דבר מגיעים ל-**root CA** שהוא <u>הישות הראשית ממנה מושרש האמון בתקן</u>.

בהמשך המסמך ננתח את תוכן התעודות על כלל רבדיו השונים, ניצר תעודות ונציג פרוטוקולים העושים שימוש נרחב בתעודות.

קבלת תעודה ב-X.509

בסדר, תעודות תעודות אבל מה אני עושה איתן ומאיפה אני מקבל אותן בכלל...? תקן X.509 מגדיר 2 דרכים לקבלת סרטיפיקט:

תצורת Central: בשיטה זו ה-CA (ה-server) מייצר את המפתח הפרטי והציבורי עבור הלקוח (subject) ושולח אותם (בצורה מאובטחת) ללקוח. שיטה זו מתאימה לחלוקה של תעודות שמאוחסנות על אמצעי פיזי ממוגן כזה או אחר כדוגמת smart card.

תצורת Distributed: בשיטה זו ה-client מייצר בעצמו את זוג המפתחות שלו ושולח בקשה אל ה-CA עם המפתח הציבורי שלו. ה-CA מחזיר לו את הסרטיפיקט עבור המפתח הציבורי. חשוב לציין כי בשיטה הזו על אחריות ה-CA לוודא כי הלקוח אכן יודע ומחזיק במפתח הפרטי המתאים. בהמשך המסמך נממש תצורה זו.

פרוטוקולים התומכים ב-X.509

תקן X.509 הוא תקן חשוב בתחום אבטחת המידע מכיוון שפרוטוקולים רבים נשענים על סרטיפיקטים ופרוטוקולי אימות של X.509 (במידה וקיימת בו פרצת אבטחה פלטפורמות אחרות העושות בו שימוש יהיו פגיעות גם הן). פרוטוקולים נפוצים העושים שימוש ב-X.509v3:

TLS/SSL	XMPP	S/MIME
HTTPS	IPSec	EAP
	SSH	LDAP

כל פרוטוקול העושה שימוש ב-TLS ו\או HTTPS יורש את השימוש בתעודות X.509 מכיוון שהן הסטנדרט - פרוטוקולים כדוגמת SMTP, IMAP ,POP, LDAP, XMPP ועוד רבים. מכשירים כגון כרטיסים חכמים ו-TPMs לרוב מחזיקים תעודות X.509.

SSL vs TLS

אנקדוטה קטנה שאני בטוח שרובכם מכירים כבר אבל רק כמה מילים על ההבדל בין SSL ל-TLS ברשותכם:

SSL (Secure Socket Layer) היה הפרוטוקול המקורי בו נעשה שימוש להענקת הצפנה לתעבורת HTTP, בתצורת HTTPS. היו 2 הפצות של SSL - גרסה 2 וגרסה 3. שתי הגרסאות נמצאו לקויות מבחינת אבטחה קריפטוגרפית ולכן **מומלץ לנטוש כל שימוש בהן**. עקב סיבות פיננסיות ומיתוג, גרסה 3.1 של SSL קיבלה את השם היוקרתי החדש TLS (Transport Layer Security) גרסה 1.0 ובהתאם, עם הזמן יצאו גרסאות 1.1, 1.2, 1.3 של פרוטוקול TLS.

השימוש בשמות "SSL/TLS", "SSL" ו-"TLS" לרוב נעשה בצורה דו-כיוונית ובא במטרה לתאר את אותו פרוטוקול, במסמך אני אשתמש ב-"TLS" בלבד וכאשר אזכיר את "SSL" זה יהיה למטרת רפרנס לפרוטוקול legacy בלבד.

SSL 3.1==TLS 1.0

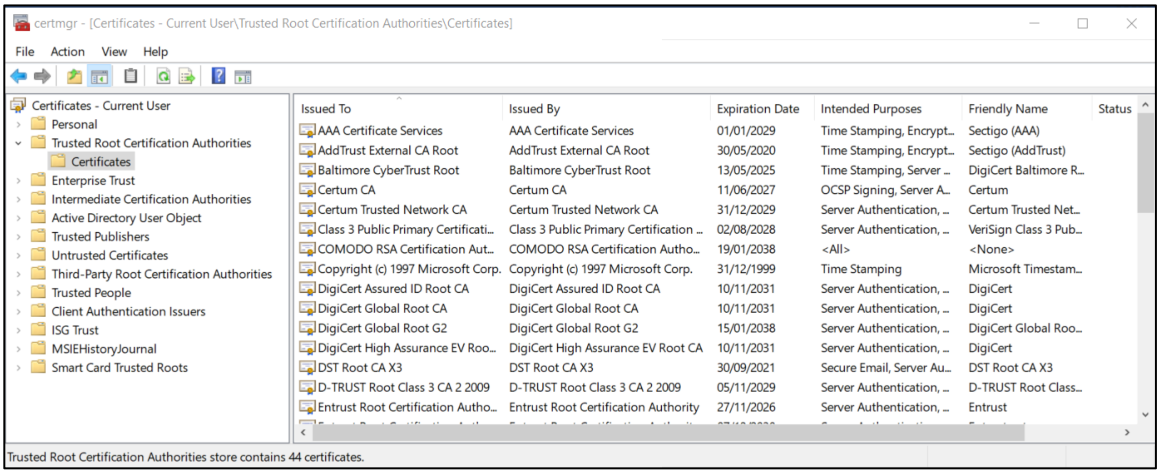
מהיכן מגיעות certificates למחשב?

ציינו שעל מנת לאמת את החתימה הדיגיטלית של התעודה עלינו לפתוח אותה בעזרת המפתח הציבורי של ה-CA, שאלה אינטואיטיבית שעולה מהצורך הזה היא מניין לנו המפתח הציבורי של ה-CA? כיצד אנו מקבלים אותו בצורה בטוחה ולא "זיוף שלו"? מהיכן בעצם מגיעות התעודות שלנו? מי מנהל אותן? המון שאלות צצות במהרה ובכללי אפשר לחלקן ל-2 תתי שאלות עיקריות:

- מהיכן מגיעות **Server** Certificates למחשב
- מהיכן מגיעות **Client** Certificates למחשב

מהיכן מגיעות Server Certificates למחשב?

ובכן, כל הדפדפנים המודרניים מגיעים בהתקנה עם רשימה של Certificates של ה-CAs המרכזיים. במערכת ההפעלה Windows קיים **Certificate Store** והוא אזור אחסון (ומוצפן) שמכיל תעודות שמגיעות עם מערכת ההפעלה. כאשר אנחנו מתקינים או מעדכנים את הדפדפן - הוא מוסיף ומעדכן את התעודות ב-Certificate Store. ולכן, אפשר להבין שזוהי ההחלטה של כל דפדפן על אלו CAs לסמוך.



ניתן לצפות ב-Certificate Store בעזרת הפעלה של *certmgr.msc* מ-CMD / (*winkey+R*).

Root CAs בדרך כלל חותמים דיגיטלית על עצמם - כלומר בעזרת ה-public key שבתעודה ניתן לאמת את החתימה הדיגיטלית שמופיעה על התעודה. CAs משניים (Intermediate) נחתמים על ידי Root CAs.
כמו כן, חייבים להזכיר כי קיימת אופציה לצרוב ידנית תעודות שרת / לקוח למחשב ובכך לספק לכל תוכנה / שירות אמצעי הזדהות מוכן.

מהיכן מגיעות Client Certificates למחשב?

האם איי פעם נפגשתם עם סוכן של Go Daddy או VeriSign ונדרשתם להוכיח את זהותכם? - אני מניח שלא. יש כמה דרכים לייצר Client Certificate:

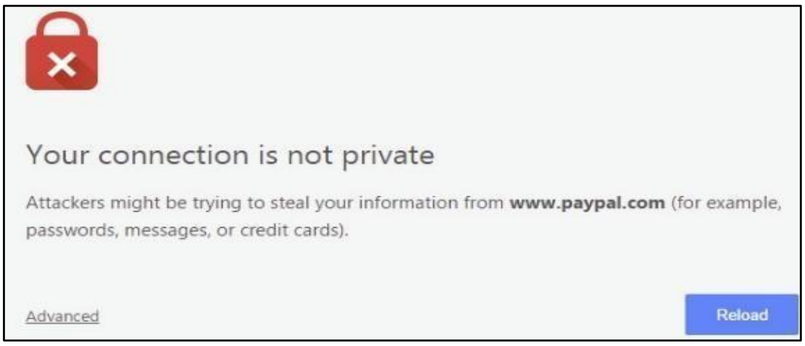
- ליצור לוקאלית את המפתח ולחתום על התעודה בעצמנו. בפרק מתקדם יותר נדגים כיצד ניתן לעשות את כל התהליך של יצירת מפתח, השמתו לקובץ certificate על בסיס המפתח ולאחר מכן טעינת התעודה לשרת.
- השרת מייצר certificate עבורנו וחותם עליו. לאחר מכן, כל מה שצריך לעשות זה להוריד את הקובץ ולעשות לו import לתוך ה-Certificate Store.
- דרך אחרונה ומקובלת היא שארגון ה-IT שאליו אנחנו משויכים, יוצר את ה-certificate ומשתיל אותו בהתקנה / בסקריפט שרץ בהפעלת המחשב / מעביר אותו בצורה קשיחה (על גבי USB או דיסק). כמובן שדרך זיהוי זו תקפה בעיקר לאימות (Authentication) פנים-ארגוני ולא למערכות אינטרנט.

כל מועדון צריך סלקטור CRL - -Certificate Revocation List

כדי לזהות מקרים בהם נעשה שימוש לא מורשה בתעודות שנגנבו או אבדו, ישות CA מחזיקה רשימה של כלל התעודות שהונפקו על ידה ושבוטלו באופן יזום, **לפני תום תוקפן**.

אנאלוגיה שאפשר למשוך ממנה קווי דמיון היא העובדה שחברות אשראי מחזיקות רשימות אשר מכילות את מספרי כרטיס האשראי שבוטלו עקב גניבה או אובדן, על מנת למנוע הונאות בכרטיסי אשראי. כאשר מתבצעת רכישה המערכת בודקת בצורה אוטומטית האם כרטיס האשראי אכן בתוקף והוא לא נמצא ברשימה של כרטיסי האשראי אשר נגנזו.

בצורה דומה, דפדפן הלקוח מבצע פניה אוטומטית לרשימת התעודות הבטלות בכדי לוודא כי אכן התעודה של האתר לא בוטלה. הרשימה מתעדכנת כל 12 שעות באופן קבוע, והיא תקפה ל-24 השעות הבאות. במידה והתעודה מופיעה כבטלה, חיווי התעודה יהיה לא תקין והדפדפן יתריע על כך:

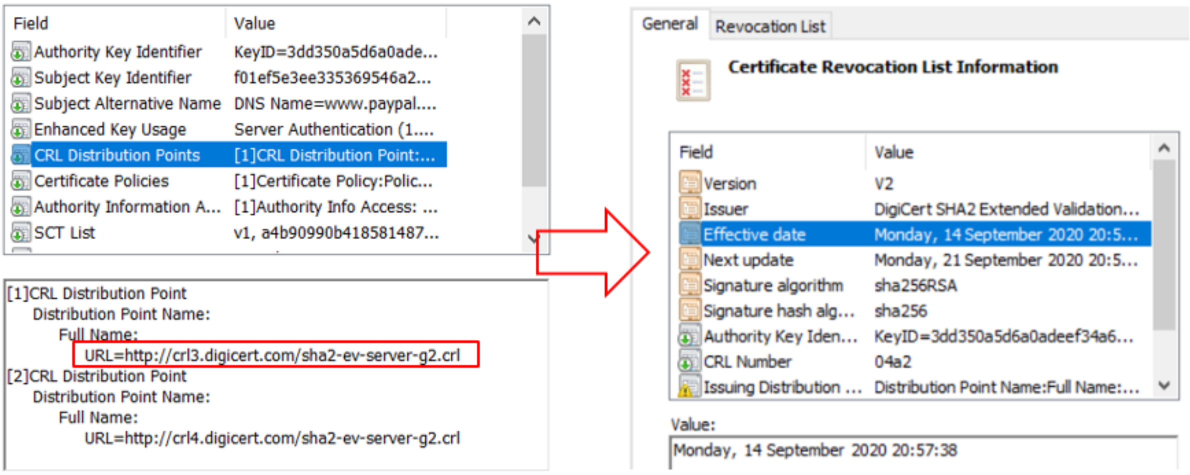


CRL חתומה בעצמה על ידי CA ופרט לרשימת הסרטיפיקטים המבוטלים, היא מכילה מספר סידורי, תאריך פרסום הרשימה והתאריך בו תפורסם הרשימה הבאה.

איך לפתוח CRL?

uno - לפתוח תעודה כמו שעשינו בהתחלה ולחפש את השדה CRL Distribution Points

dos - בשדה יהיה לינק להורדת CRL עדכני, כל מה שצריך זה פשוט להעתיק אותו לשורת ה-URL:



ישנם 2 סוגים שונים של ביטול תעודות כפי שמתועד ב-RFC 5280:

ביטול (Revoked) - ביטול התעודה באופן **בלתי הפיך**. כאשר למשל התגלה כי ה-CA הנפיק לא כמו שצריך את התעודה, או אם קיים חשד מעל סביר כי האבטחה של המפתח הפרטי של מקבל התעודה נפגעה. תרחיש נוסף הוא גילוי בדיעבד כי מקבל התעודה לא עמד בדרישות המדיניות של ה-CA. מקרים בהם התגלה כגון פרסום מסמכים כוזבים, מצג שווא של התנהגות תוכנה או הפרה של מדיניות אחרת שצוינה על ידי מפעיל CA או לקוחה. הסיבה הנפוצה ביותר לביטול היא שהמשתמש כבר לא נמצא ברשותו הבלעדית של המפתח הפרטי (למשל, ה-token שמכיל את המפתח הפרטי אבד או נגנב).

החזקה זמנית (Hold) - ניתן להשתמש במצב **הפיך** כדי לציין את תקופת האישור הזמנית (למשל, תקופה קצרה בה המשתמש לא בטוח אם המפתח הפרטי שלו אבד). אם המפתח הפרטי נמצא לאחר הכנסת התעודה למצב Hold ונמצאו ההוכחות כי לאיש לא הייתה גישה אליו, ניתן יהיה להחזיר את הסטטוס לתעודה ולתוקף ובכך להסיר את האישור מה-CRL שעתיד להתפרסם במחזור הבא.

השימוש ב-CRL הופך את תהליך האימות למסובך. כמה מסובך? ובכן, גודל הרשימות יכול להיות מאוד גדול, מה שמשפיע בצורה ישירה על הזמן הנדרש לאימות התעודה. פתרון חלקי לבעיה הוא לחלק את ה-CRL למספר תתי רשימות.

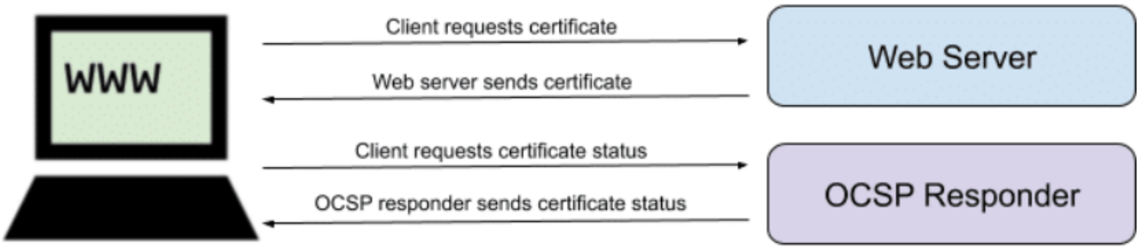
ישנם פתרונות נוספים כמו השימוש ב-OCSP אותו נראה בקרוב, אך חשוב להבין כי תמיד יהיה tradeoff בין אורך תהליך האימות של התעודה נתונה ואיכותו של מנגנון ה-revocation.

Online Certificate Status Protocol - OCSP

OCSP הוא פרוטוקול המשמש את דפדפני האינטרנט לקביעת סטטוס הביטול של תעודות TLS המסופקות על ידי אתר HTTPS **בזמן החיבור**.

בתצורתו הפשוטה ביותר OCSP פועל באופן הבא:

1. דפדפן אינטרנט מקבל תעודה מאתר המריץ HTTPS.
2. דפדפן האינטרנט שולח בקשה ל-OCSP responder - שרת המופעל על ידי ה-CA שהנפיק את האישור.
3. התגובה החתומה של ה-OCSP responder לדפדפן מציינת אם האישור תקף או בוטל.



על פניו, הלוגיקה נראית טוב - אני, בתור לקוח מן המניין, מתשאל בזמן אמת נציג של ה-CA אם התעודה שקיבלתי טובה או לא. אבל זה לא המצב בשטח.

בפועל, OCSP חושף את כלל המערכת לבעיות רבות. האחת, והברורה מכולן - **זמן חישוב נוסף**. על מנת שאותו OCSP responder יחפש במסד הנתונים האם התעודה וולידית או לא הוא צריך זמן. על פי דיווחים של Cloudflare, זה יכול להפוך את TLS לאיטי יותר ב-33%. בנוסף, זמן החציון הממוצע של Successful OCSP check הוא ~300ms כאשר זמן התוחלת מתקרב לשנייה - מדובר בהמון זמן. שנייה בהייה במסך לבן היא זמן לא מבוטל ולכן זה לא מפתיע שדווח כי הרבה משתמשים יוצאים מהאתר.

כתוצאה מכך, דפדפנים חסרי סבלנות החלו ליישם OCSP check בתצורה של **soft-fail**. כלומר, אם שרת OCSP לא נגיש בזמן סביר, התעודה תיחשב וולידית והדפדפן ימשיך עם חיבור ה-HTTPS.

התקפות MITM ניצלו את אופן הפעולה הנ"ל בכך שהם חסמו את כל OCSP queries והשתמשו בתעודות ישנות בשביל להזדהות כשירות HTTPS אמין.

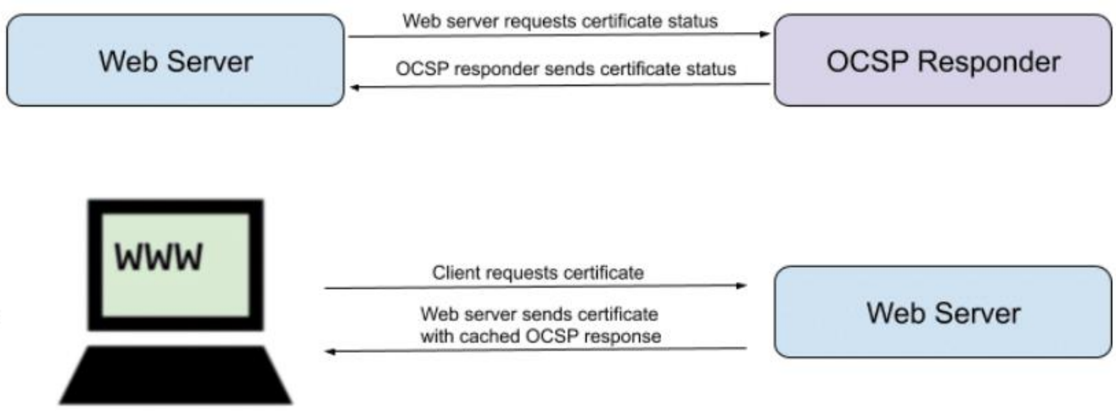
בעיה שנייה היא **פרטיות**. כפועל יוצא מהפנייה של הדפדפן ל-OCSP responder, ה-CA לומד את כתובת ה-IP של המשתמש ובאיזו אתרים הוא מבקר.

בניסיון לפתור את הבעיות החדשות שהציב OCSP הוצע פתרון חדש - OCSP Stapling.

OCSP Stapling

אופן פעולתו:

1. שרת רשת (המריץ יישום) מגיש בקשה מבעוד מועד ל-signed OCSP response עבור כל התעודות שלו מ-OCSP responder. אישורים אלו יכולים **להישמר בזיכרון המטמון** לכל היותר 7 ימים.
2. שרת האינטרנט שומר את OCSP response ביחד עם התעודה ושולח אותם יחדיו ב- HTTPS responses לדפדפנים.
3. על מנת למנוע מתוקף שמנהל שרת לשלוח תעודות שנגנזו בלי החתימה של ה-OCSP response עליהן, נוספה הרחבה לתעודות הללו אשר מכריחות חתימה של OCSP.



אני סבור שלא תופתעו לשמוע לגם ב-OCSP Stapling התגלו פערים אבטחתיים וכי גם הוא לא קרוב להוות פתרון מושלם.

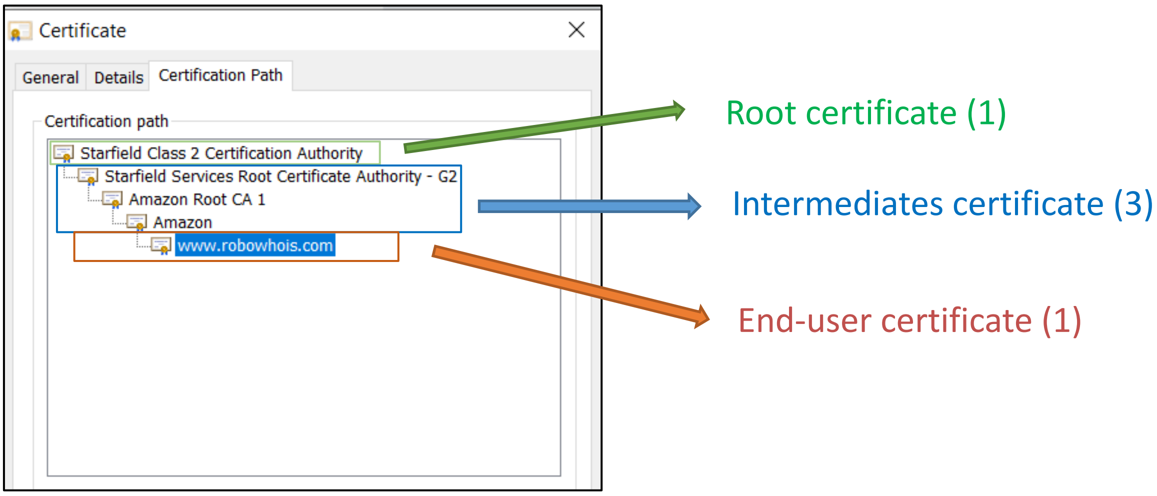
בהינתן כל הנ"ל קשה לומר ש-OCSP אכן מהווה פתרון של ממש, עם זאת, **חשוב להדגיש כי החסרונות העיקריים שהוצגו בתת פרק האחרון מתייחסים בעיקר למימוש OCSP ברשתות ומכשירים ציבוריים.**

במידה וקיים רצון למימוש שירותי web ברשת פנימית\ פרטית, OCSP מצטייר כשימושי. פרטיות היא פחות פונקציה כשמדובר על מערכת פרטית ומוכרת, כמו כן, ניתן להרים שירות OSCP responder מהיר מספיק ולהשתמש בתצורה של hard-fail לשירותי web.

סומך עלייך אח שלי - Certificate Chain

ישנם 2 סוגים של CAs - **root CA** ו- **intermediate CA.** עקב סיבות ניהוליות ואבטחתיות, תעודות X.509 לרוב יגיעו משולבות לכדי שרשרת של אימותים. הרעיון הוא פשוט - CA יכול לחתום על תעודה של CA אחר (כמובן לאחר אימות ודאי) ובכך לאפשר לקהל משתמשיו שימוש באותו CA.

רשת תעודות (certificate chain) או היררכיית תעודות (certification path) היא רשימה של תעודות המתחילה בתעודת היעד המכילה את פרטי נושא התעודה, דרך תעודות ביניים ועד לתעודת השורש:

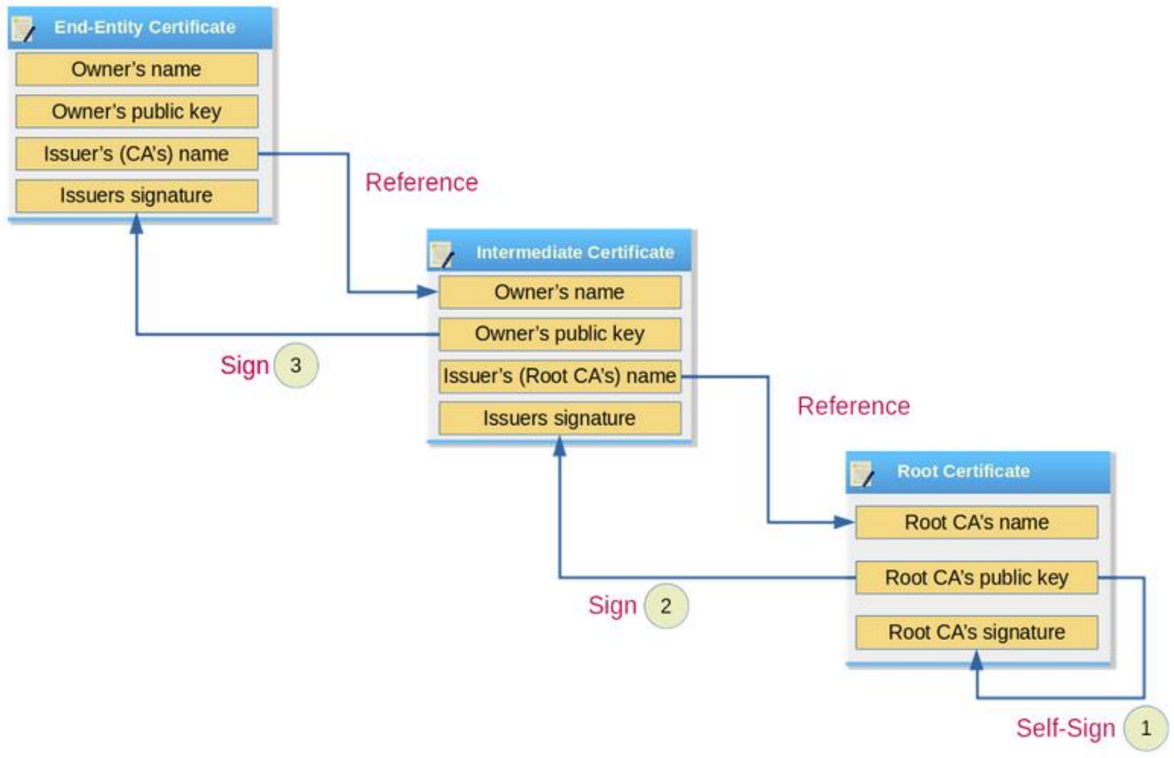


כאשר מכשיר / לקוח קצה (הדפדפן לדוגמה) ניגש לאמת תעודה הוא ראשית בודק האם הוא סומך על ה-CA שהנפיק אותה. במידה והוא לא סומך עליו, הוא יבדוק האם התעודה של אותו CA הונפקה על ידי CA שעליו הוא כן סומך. כך למעשה מתבצעת בדיקה רקורסיבית (רק מחשבתית, בפועל מטעמי אבטחה, אסור שתתבצע בדיקה רקורסיבית) עד אשר הלקוח מוצא תעודה שהנפיק CA עליו הוא סומך (משמע הוא יכול לסמוך על תעודת היעד). במידה והבדיקה הגיעה לסוף השרשרת, והלקוח לא סומך על ה-root CA הוא לא יאשר את החיבור והודעת שגיאה תקפוץ לגוף הדפדפן. מגניב? מגניב.

המאפיינים של שרשרת תעודות:

1. שם המנפיקה של כל תעודה למעט האחרונה תואם את שם הנושא בתעודה הקודמת (כל תעודה מאמתת את בעלי התעודה הקודמת בשרשרת).
2. כל תעודה (למעט האחרונה) חתומה עם המפתח הפרטי של הישות ששמה מופיע בתעודה הבאה אחריה בשרשרת. כלומר את החתימה המופיעה על תעודה אחת אפשר לאמת על ידי המפתח הציבורי שמופיע בתעודה הבאה אחריה.
3. תעודת השורש (התעודה האחרונה) חותמת על עצמה.

תהליך בניית שרשרת התעודות מפורט ב-RFC 4158.



Publicly trusted vs Privately trusted PKI

זוכרים שדיברנו PKI ואיך ש-X.509 ממש אותו? אז קיימים לו 2 קונפיגורציות ראשיות. אם כי 2 הקונפיגורציות מספקות את אותה מטרה, השוני ביניהן ניכר. תוכנת לקוח סומכת על PKIs ציבוריים באופן אוטומטי, ואילו על PKIs פרטיים סומכים רק לאחר שהמשתמש אישר לסמוך עליהם בצורה ידנית והתקין את התעודות שה-PKI שלח. ארגון המקיים PKI בעל אמון ציבורי הוא CA.

למה לבחור ב-Publicly trusted PKI?

במידה ואנחנו מתכננים להרים אתר\ מערכת מקוונת **וציבורית**, שימוש ב-publicly trusted PKI תמיד יהיה הפתרון מכיוון שלדרוש מכל מבקר להתקין ידנית את תעודות השורש של האתר שלנו זה פשוט לא ריאלי ופרקטי.

למה לבחור ב-Privately trusted PKI?

Public PKI נדרשים לעקוב באדיקות אחר רגולציות ולעבור ביקורות תקופתיות ואילו Private PKI ראשי **לוותר על כל הדרישות** ולסטות מהסטנדרטים (כגון אלו של X.509) בכל דרך שהוא רואה לנכון. למרות שהדבר יכול לגרום לכך ש-PKI פרטיים לא יפעלו בהתאם לשיטות העבודה המומלצות, זה מאפשר ללקוחות חופש רב יותר ביחס למדיניות התעודות ולפעולותיהם.

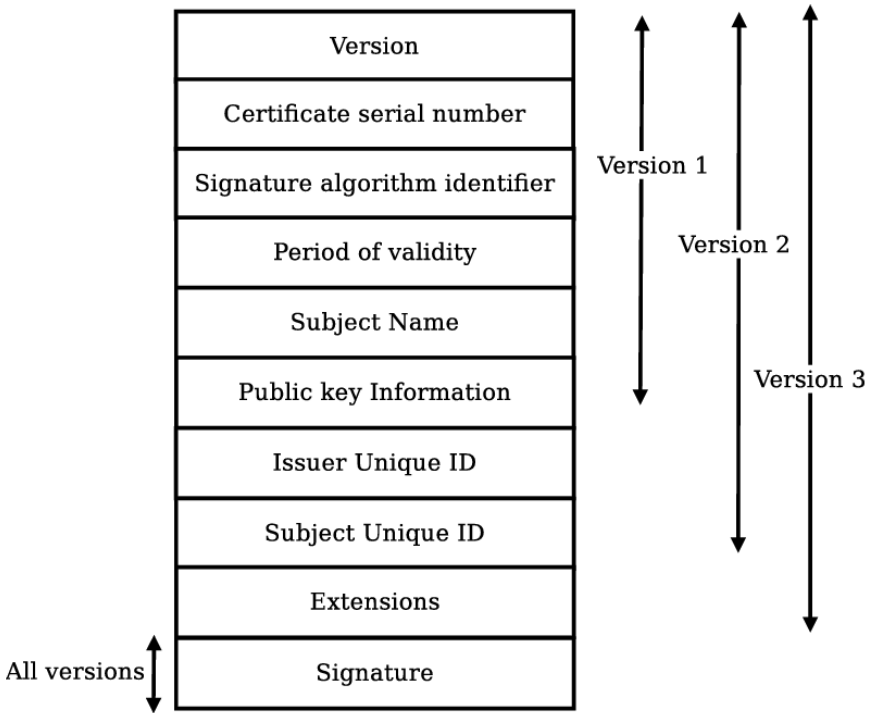
דוגמא טובה לכך היא שה-baseline של PKI ציבורי אוסר על CAs להנפיק תעודות עבור internal domains (למשל example.local) ו-PKI פרטי יכול להנפיק תעודות לכל מי שהוא רוצה, כולל דומיינים פנימיים.

תעודות של PKI ציבורי תמיד יהיו חייבים לעקוב אחרי הסטנדרט של פרופיל מיפוי לתקן X.509 המקובל לתעודות ציבוריות. עם זאת, לקוחות ספציפיים עשויים לדרוש פרופיל תעודות מיוחד. מלבד התעודה עצמה, PKI פרטי מאפשר שליטה מלאה גם על נהלי אימות הזהות ונתוניהם. ניתן לשלב מערכות בקרת גישה של הלקוח עצמו (כגון single sign-on או ספריות LDAP) עם שירות ה-PKI הפרטי כדי לספק בקלות תעודות לגורמים שכבר נותנים אמון על ידי המפעיל.

איך נראית תעודה דיגיטלית

עד כה, מהלך המסמך תוכנן לספק מידע כללי וסקירה טכנולוגית על תעודות דיגיטליות. הפרק הקרוב נכתב כאשר מטרותיו העיקריות לספק מידע שימושי ופרקטי.

כאמור, תעודה היא מבנה נתונים חתום דיגיטלית שקושר מפתח ציבורי לישות. מאז הפצתו ב-1998, פותחו 3 גרסאות של תעודות X.509. כפי שמוצג באיור הבא, כל גרסה של מבנה הנתונים שמרה על השדות שהיו קיימים בגרסה הקודמות והוסיפה עוד:



מושגים מרכזיים בתעודה דיגיטלית

חתימה דיגיטלית

מכיוון שחתימה דיגיטלית היא חלק בלתי נפרד מתקן X.509 ונמצאת על כל תעודה, יש צורך להסביר בצורה בסיסית את הרעיון העומד מאחוריה. מי שמרגיש מספיק בטוח בנושא יכול לדלג לתת-פרק הבא בכיף.

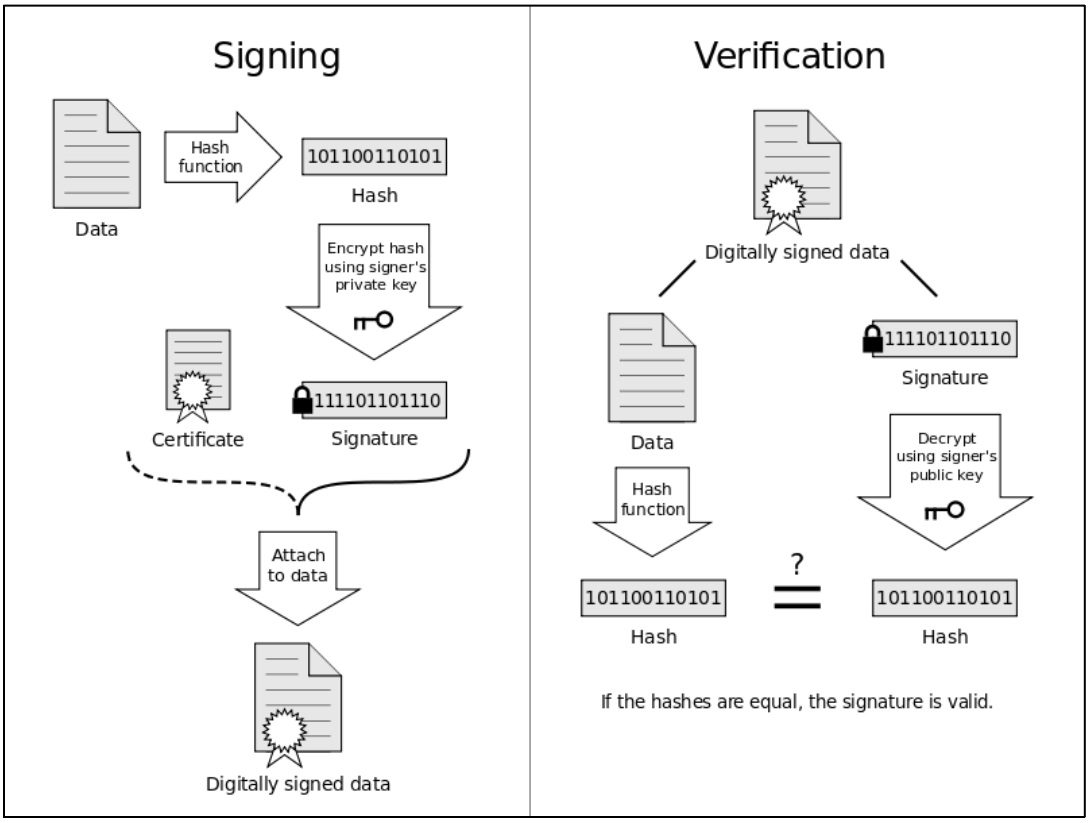
חתימה דיגיטלית היא שיטה מתמטית המאפשרת להוכיח אותנטיות של מסמך או מסר דיגיטלי ולשמור על שלמותו (מבלי לשנות אותו). בנוסף, החתימה מאפשרת מניעת הכחשה ע"י גורם שלישי מהימן וחותם זמן.

עקרונית אפשר להכין חתימה דיגיטלית באמצעות כל שיטת הצפנה אסימטרית, כאשר **המפתח הפרטי משמש לחתימה והמפתח הציבורי משמש לאימות.**

לצורך החתימה יש להשתמש בזוג מפתחות **שונה** מזה של ההצפנה.

מסיבות של יעילות מקובל ליישם את פונקציית החתימה על ערך גיבוב (hash) - פונקציה חד כיוונית שמתמצתת את המסמך במקום את המסמך עצמו. כלומר תחילה מפיקים מהמסמך ערך ייחודי קטן באמצעות hash ועל ערך זה חותמים.

<u>תרשים פשוט שעוזר לקשור את רעיון החתימה לתעודה דיגיטלית:</u>



ASN.1

פורמט ה-**ASN.1** (X.680) משמש לפריסה של תעודה X.509v3 ולכן חשוב לתת לו את הבמה לפני שנכנסים לניתוח גוף התעודה.

ASN.1 (Abstract Syntax Notation number One) הוא שפת תיאור ממשק סטנדרטית (schema language) להגדרת מבני נתונים הניתנים לפעולות serialized & deserialized בתצורה cross-platform.

עושים שימוש ב-ASN.1 באופן נרחב בטלקומוניקציה (telecommunication), ברשתות מחשבים ובקריפטוגרפיה.

יתרון של ASN.1 הוא העובדה שקידוד הנתונים עצמו אינו תלוי במחשב או בשפת תכנות מסוימת והעובדה שהוא ניתן לקריאה אנושית וקריאת מכונה.

אחת הסיבות העיקריות לפופולריות של ASN.1 היא העובדה שהוא כולל מספר סטנדרטים של שיטות קידוד כגון **BER** (Basic Encoding Rules) ו- **PER** (Packed Encoding Rules), אשר נמצאים בשימוש רב ליישומים הנאלצים לעבוד **ברוחב פס צר**. מתודות אלו מתארים כיצד יש לקודד את הערכים המוגדרים ב - ASN.1- לשידור (כלומר כיצד ניתן לתרגם אותם ל -'bytes over the wire' והפוך). כזכור, ללא קשר למכונה, לשפת התכנות, או כיצד היא מיוצגת ביישום עצמו.

הערה לכל הגיבורים שבנינו: ASN.1 מסובך ומומלץ להשתמש בקומפיילר ייעודי במידה ונדרשים לבנות תעודה תקנית.

דוגמא לסכמת ASN.1 אל מול מידע המיוצגת ב-JSON:

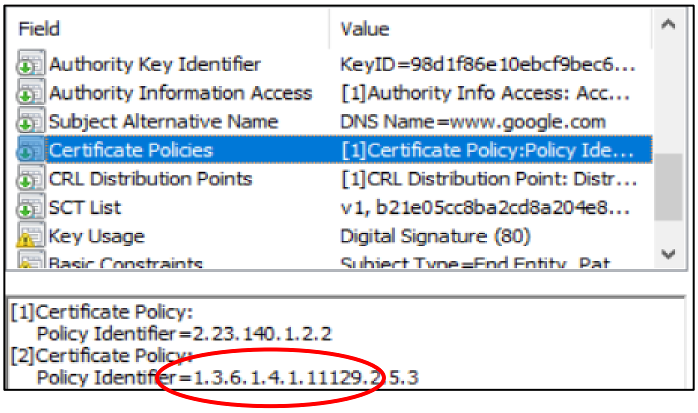
```
GeneratedSchema DEFINITIONS AUTOMATIC TAGS
::= BEGIN
GeneratedType ::= SEQUENCE {
   name UTF8String,
   owner UTF8String,
   areWeInMars BOOLEAN,
   fuel UTF8String,
   speed SEQUENCE {
      mph INTEGER
   },
   payload SEQUENCE OF UTF8String
}
END
```

```
{
  "name":"Falcone",
  "owner":"Elon Musk",
  "areWeInMars":false,
  "fuel":"liquid",
  "speed":{
    "mph":18000
  },
  "payload":["Car", "GPS unit"]
}
```

ממשק ה-API לרישום תעודות עושה שימוש ב-ASN.1 בשביל להגדיר, לקודד ולפענח certificate requests ותעודות שמועברות בין מחשבי client ו -CA. בסעיף הבא נסקור תעודה וחלקים מתעודה שמיוצגים באמצעות ASN.1.

OID - Object Identifier - כאילו החיים לא היו מסובכים גם ככה

כשמנתחים שדות בתעודה דיגיטלית נתקלים לא פעם בשדה המכיל ערך "מוזר" שנראה בצורה הבאה:



מדובר ב-OID:

OID הוא מנגנון זיהוי סטנדרטי שנוצר על ידי ITU ו-ISO על מנת לתת "שם" **ייחודי** לכל אובייקט, דבר או רעיון. כן כן, כמעט כל רעיון שקשור ברשתות, IT, אבטחה ושטויות נוספות נוכח שם. הוא מוגדר פורמלית ב-X.660. תקן PKIX ממליץ לכל האובייקטים, כגון extensions והצהרות, המשמשים בתעודות להיכלל בצורה של OID. בצורה כזו, משתפרת היכולת של פעולה הדדית בין ארגונים ברשת משותפת מכיוון שהם דוברים את אותה "שפה".

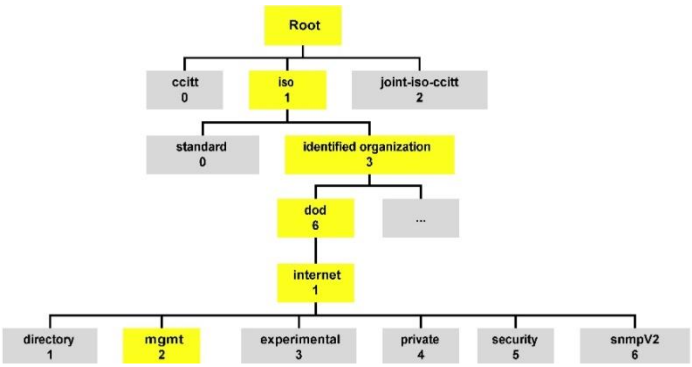
OID tree מכיל מספר צמתים (nodes) אשר כל אחת בתורה מחלקת את העץ לתתי-אובייקטים שונים. שורשי העץ מכילים את הספרות 0,1,2 אשר מסמלים את הגוף אליו האובייקט משתייך:

- 0: ITU-T
- 1: ISO
- 2: joint-iso-itu-t

כל צומת בעץ מיוצג על ידי סדרת מספרים שלמים המופרדים על ידי נקודות, המתאימות לנתיב מהשורש דרך סדרת צמתי אבות לצומת. כך למשל, ה-OID של Intel Corporation הוא: 1.3.6.1.4.1.343

כאשר ה"מסלול" דרך ה-OID tree הוא כדלקמן:

- **1** ISO
- **1.3** identified-organization,
- **1.3.6** dod,
- **1.3.6.1** internet,
- **1.3.6.1.4** private,
- **1.3.6.1.4.1** IANA enterprise numbers,
- **1.3.6.1.4.1.343** Intel Corporation



ייצוג טקסטואלי של מסלול OID בדוגמא הוא:

- **iso.identified-organization.dod.internet.private.enterprise.intel**

DB של OID אפשר למצוא ב-[Global OID reference database](#).

כאשר עוברים על גבי מסלול OID יש פירוט על כל הצמתים אליהן אפשר לגשת בהמשך ומה הן מייצגות. למשל אם נמצאים בצומת 2.5.4. נוכל להסתכל על כל ה-98 הצמתים שנמצאים מתחתיה (2.5.4.0-2.5.4.97):

Reference record for OID 2.5.4

2 joint-iso-itu-t, joint-iso-ccitt > 5 ds > 4 attributeType

parent

2.5 (ds)

node code

4

node name

attributeType

dot oid

2.5.4

asn1 oid

- {joint-iso-itu-t(2) ds(5) attributeType(4)}
- {joint-iso-ccitt(2) ds(5) attributeType(4)}

iri oid

- /joint-iso-itu-t/ds/attributeType
- /joint-iso-ccitt/ds/attributeType

iri by oid_info

/Joint-ISO-ITU-T/5/4

First Registration Authority (recovered by parent 2.5)

Hoyt Kesterson & R. Exner

Current Registration Authority (recovered by parent 2)

ITU-T SG 17 & ISO/IEC JTC 1/SC 6

Children (98)

OID	Name	Sub children	Sub Nodes Total	Description
2.5.4.0	objectClass	0	0	Object classes
2.5.4.1	aliasedEntryName	1	1	Attribute type "Aliased entry name"
2.5.4.2	knowledgeInformation	0	0	knowledgeInformation attribute type
2.5.4.3	commonName	1	1	Common name
2.5.4.4	surname	1	1	Attribute "surname"
2.5.4.5	serialNumber	1	1	Serial number attribute type
2.5.4.6	countryName	1	1	Country name
2.5.4.7	localityName	2	3	Locality Name
2.5.4.8	stateOrProvinceName	2	3	State or Province name
2.5.4.9	streetAddress	2	3	Street address
2.5.4.10	organizationName	2	3	Organization name
2.5.4.11	organizationUnitName	2	3	Organization unit name
2.5.4.12	title	1	1	Title attribute type
2.5.4.13	description	1	1	Description attribute type
2.5.4.14	searchGuide	1	1	Search guide attribute type

שדות בתעודה דיגיטלית (אני הייתי שוקל להרתיח מים לכוס נוספת ☕)

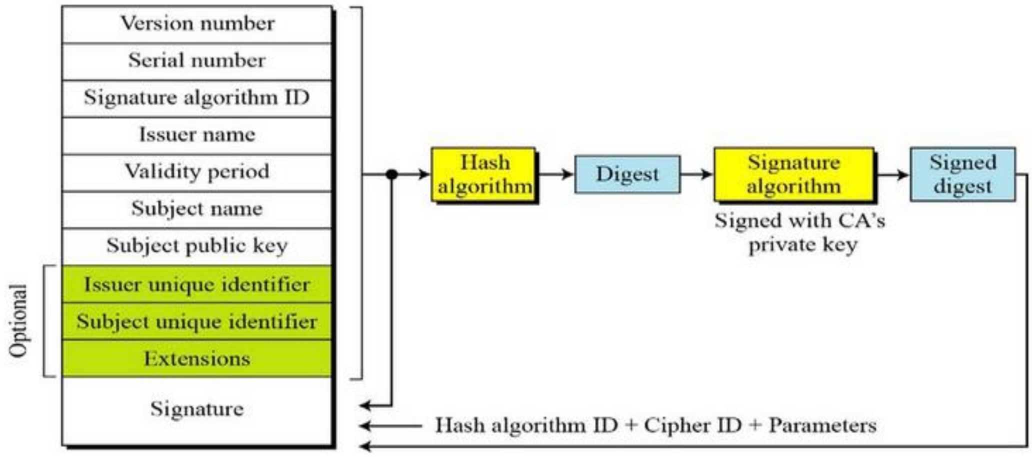
פורמט כללי של X.509

ישנן 3 רמות של אבסטרקציה:

1. ההבנה הכללית של איך בנויה התעודה
2. כיצד התעודה נראית במחשב כשפותחים אותה
3. התוכן שמועבר "באמת" - בפורמט ASN.1.

קודם כל נעבור על רמות האבסטרקציה השונות ולאחר מכן, כשנכיר את מבנה התעודה, נרחיב על כל אחד מהשדות.

כיצד בנויה תעודה בכללי:



אלו השדות הכלולים בכל תעודה. גוף התעודה מורכב מכל השדות ביחד עם חתימת ה-CA שערב לתעודה. כפי שניתן לראות באיור, גוף התעודה נכנס למטודת גיבוב וה-CA חותם על ה-hash עם המפתח הפרטי שלו ומוסיף את החתימה לתעודה.

כיצד נראית תעודה בממשק המשתמש של Windows:

Field	Value
Version	V3
Serial number	3be40f3bf64186b602000000...
Signature algorithm	sha256RSA
Signature hash algorithm	sha256
Issuer	GTS CA 1O1, Google Trust Ser...
Valid from	Tuesday, 5 May 2020 11:31:24
Valid to	Tuesday, 28 July 2020 11:31:24
Subject	www.google.com, Google LLC,...
Public key	ECC (256 Bits)
Public key parameters	ECDSA_P256
Enhanced Key Usage	Server Authentication (1.3.6....
Subject Key Identifier	b6a174e61186547cd9291e70f...
Authority Key Identifier	KeyID=98d1f86e10ebcf9bec6...
Authority Information Access	[1]Authority Info Access: Acc...
Subject Alternative Name	DNS Name=www.google.com
Certificate Policies	[1]Certificate Policy:Policy Ide...
CRL Distribution Points	[1]CRL Distribution Point: Distr...
SCT List	v1, b21e05cc8ba2cd8a204e8...
Key Usage	Digital Signature (80)
Basic Constraints	Subject Type=End Entity, Pat...
Thumbprint	95e28236e041a6fa8e538c188...

כיצד נראית התעודה בפורמט ASN.1:

```
Certificate ::= INTERGER
{
    tbsCertificate          TBSCertificate,
    signatureAlgorithm      Object Identifier,
    signature               BITSTRING
}
```

ככה בעצם נראית התעודה - סדרה רצופה של אובייקטים. האובייקט הראשון מכיל את תוכן העניין שלנו. הוא נקרא To Be Signed Certificate- **tbsCertificate**.

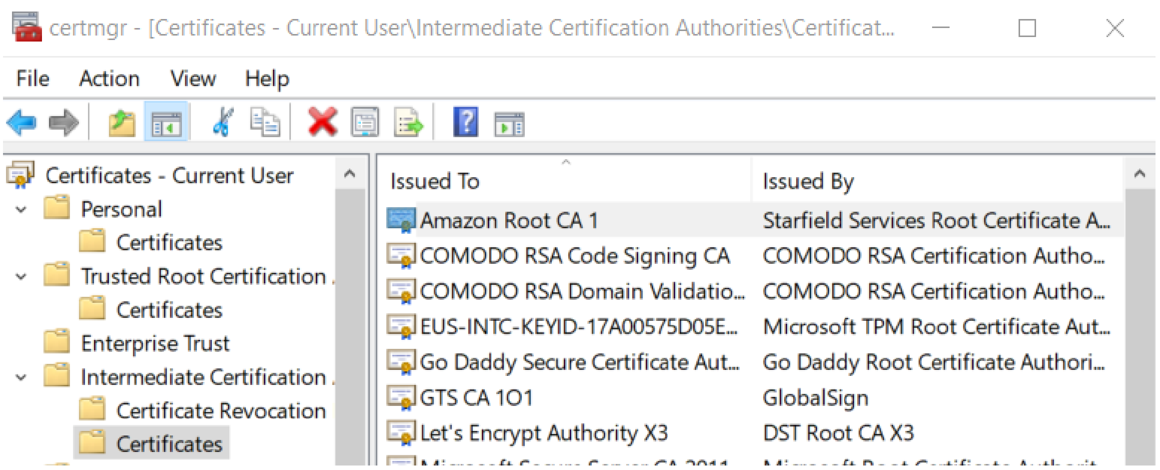
האובייקט השני, **signatureAlgorithm**, הוא סוג החתימה שה-CA השתמש בשביל לחתום על התעודה (למשל SHA256).

השדה השלישי, **signature**, הוא לא אובייקט אלא רצף של סיביות שקשורים לחתימה של ה - TBSCertificate אחרי קידוד.

נפשט את תוכן האובייקט **TBSCertificate** (ב -ASN.1 כל אובייקט יכול להכיל גם רשימה של אובייקטים ושדות בתוכו - אובייקט בתוך אובייקט):

```
TBSCertificate ::= SEQUENCE
{
  version                  [0] Version DEFAULT v1(0),
  serialNumber             CertificateSerialNumber,
  signature                AlgorithmIdentifier,
  issuer                   Name
  validity                 Validity,
  subject                  Name
  subjectPublicKeyInfo     SubjectPublicKeyInfo,
  issuerUniqueIdentifier   [1] IMPLICIT UniqueIdentifier OPTIONAL,
  subjectUniqueIdentifier  [2] IMPLICIT UniqueIdentifier OPTIONAL,
  extensions               [3] Extensions OPTIONAL
}
```

על מנת להמחיש כיצד תעודות נראות "באמת", נציג תעודת X.509 מפוענחת של amazon root CA (שנמצאת לוקאלית על המחשב שלנו):



תוכן התעודה המפוענחת בפורמט ASN.1 (באמצעות שימוש ב-certutil או ב-openssl):

```
X509 Certificate:
Version: 3
Serial Number: 067f944a2a27cdf3fac2ae2b01f908eeb9c4c6
Signature Algorithm:
    Algorithm ObjectId: 1.2.840.113549.1.1.11 sha256RSA
    Algorithm Parameters:
    05 00
Issuer:
    CN=Starfield Services Root Certificate Authority - G2
    O=Starfield Technologies, Inc.
    L=Scottsdale
    S=Arizona
    C=US
  Name Hash(sha1): 887da4445e67ea7c94774e43189c3ecee4c87312
  Name Hash(md5): 52135310639a10f77f886b229b9f7afc

 NotBefore: 25/05/2015 15:00
 NotAfter: 31/12/2037 4:00

Subject:
    CN=Amazon Root CA 1
    O=Amazon
    C=US
  Name Hash(sha1): 4f59a39453cfb9559e7f6bd8c54da53da642b714
  Name Hash(md5): 99c508fd54cbc396cc3256bb87829ae0

Public Key Algorithm:
    Algorithm ObjectId: 1.2.840.113549.1.1.1 RSA
    Algorithm Parameters:
    05 00
Public Key Length: 2048 bits
Public Key: UnusedBits = 0
    0000  30 82 01 0a 02 82 01 01  00 b2 78 80 71 ca 78 d5

Certificate Extensions: 7
    2.5.29.19: Flags = 1(Critical), Length = 5
    Basic Constraints
        Subject Type=CA
        Path Length Constraint=None

    2.5.29.15: Flags = 1(Critical), Length = 4
    Key Usage
        Digital Signature, Certificate Signing, Off-line CRL Signing, CRL
Signing (86)

    2.5.29.14: Flags = 0, Length = 16
    Subject Key Identifier
        8418cc8534ecbc0c94942e08599cc7b2104e0a08

    2.5.29.35: Flags = 0, Length = 18
    Authority Key Identifier
        KeyID=9c5f00dfaa01d7302b3888a2b86d4a9cf2119183

    1.3.6.1.5.5.7.1.1: Flags = 0, Length = 6c
    Authority Information Access
        [1]Authority Info Access
             Access Method=On-line Certificate Status Protocol
(1.3.6.1.5.5.7.48.1)
             Alternative Name:
                  URL=http://ocsp.rootg2.amazontrust.com
        [2]Authority Info Access
             Access Method=Certification Authority Issuer (1.3.6.1.5.5.7.48.2)
             Alternative Name:
                  URL=http://crt.rootg2.amazontrust.com/rootg2.cer
```

```
    2.5.29.31: Flags = 0, Length = 36
    CRL Distribution Points
        [1]CRL Distribution Point
             Distribution Point Name:
                  Full Name:
                       URL=http://crl.rootg2.amazontrust.com/rootg2.crl

    2.5.29.32: Flags = 0, Length = a
    Certificate Policies
        [1]Certificate Policy:
             Policy Identifier=All issuance policies

Signature Algorithm:
    Algorithm ObjectId: 1.2.840.113549.1.1.11 sha256RSA
    Algorithm Parameters:
    05 00
Signature: UnusedBits=0
    0000  18 21 9d 34 f9 c4 a9 2a  a4 51 62 0e 9a ad fa 9a


Non-root Certificate
Key Id Hash(rfc-sha1): 8418cc8534ecbc0c94942e08599cc7b2104e0a08
Key Id Hash(sha1): 408377dd675d40870de22a8905452875f236ded4
Key Id Hash(bcrypt-sha1): 4e5ce94c78768c1064dbd81ca0fb32707fab3b48
Key Id Hash(bcrypt-sha256):
ad5e3307f114c9790dc7ce83382fb91792a8c298e40b10f00b63f0540bc03c4a
Key Id Hash(md5): 94ba2a8e68434595d5aff46246c54c12
Key Id Hash(sha256):
50027ff723aaaba9e4e025f642d6600e6c93e891d896405f8a13b37b31fff1cb
Key Id Hash(pin-sha256): ++MBgDH5WGvL9Bcn5Be30cRcL0f5O+NyoXuWtQdX1aI=
Key Id Hash(pin-sha256-hex):
fbe3018031f9586bcbf41727e417b7d1c45c2f47f93be372a17b96b50757d5a2
Cert Hash(md5): e865a22aae524d26869af0448d6fd896
Cert Hash(sha1): 06b25927c42a721631c1efd9431e648fa62e1e39
Cert Hash(sha256):
87dcd4dc74640a322cd205552506d1be64f12596258096544986b4850bc72706
Signature Hash: c95f7b20f6fcd39fd3a07a2e44252423b634fdbe35e1e045d964deea626115cb
```

מדהים, אה? חכו שתראו את הדבר הבא!

אם נבחר לצפות בקובץ הבינארי של התעודה (באמצעות תוכנת HxD) בפורמט ASN.1 מקודד לפי DER נקבל:


```
30 82 04 92 30 82 03 7A A0 03 02 01 02 02 13 06 7F 94 4A 2A 27 CD F3 FA C2 AE 2B 01 F9 08 EE B9
C4 C6 30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00 30 81 98 31 0B 30 09 06 03 55 04 06 13 02 55
53 31 10 30 0E 06 03 55 04 08 13 07 41 72 69 7A 6F 6E 61 31 13 30 11 06 03 55 04 07 13 0A 53 63
6F 74 74 73 64 61 6C 65 31 25 30 23 06 03 55 04 0A 13 1C 53 74 61 72 66 69 65 6C 64 20 54 65 63
68 6E 6F 6C 6F 67 69 65 73 2C 20 49 6E 63 2E 31 3B 30 39 06 03 55 04 03 13 32 53 74 61 72 66 69
65 6C 64 20 53 65 72 76 69 63 65 73 20 52 6F 6F 74 20 43 65 72 74 69 66 69 63 61 74 65 20 41 75
74 68 6F 72 69 74 79 20 2D 20 47 32 30 1E 17 0D 31 35 30 35 32 35 31 32 30 30 30 30 5A 17 0D 33
37 31 32 33 31 30 31 30 30 30 30 5A 30 39 31 0B 30 09 06 03 55 04 06 13 02 55 53 31 0F 30 0D 06
03 55 04 0A 13 06 41 6D 61 7A 6F 6E 31 19 30 17 06 03 55 04 03 13 10 41 6D 61 7A 6F 6E 20 52 6F
6F 74 20 43 41 20 31 30 82 01 22 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00 03 82 01 0F 00 30
82 01 0A 02 82 01 01 00 B2 78 80 71 CA 78 D5 E3 71 AF 47 80 50 74 7D 6E D8 D7 88 76 F4 99 68 F7
58 21 60 F9 74 84 01 2F AC 02 2D 86 D3 A0 43 7A 4E B2 A4 D0 36 BA 01 BE 8D DB 48 C8 07 17 36 4C
F4 EE 88 23 C7 3E EB 37 F5 B5 19 F8 49 68 B0 DE D7 B9 76 38 1D 61 9E A4 FE 82 36 A5 E5 4A 56 E4
45 E1 F9 FD B4 16 FA 74 DA 9C 9B 35 39 2F FA B0 20 50 06 6C 7A D0 80 B2 A6 F9 AF EC 47 19 8F 50
38 07 DC A2 87 39 58 F8 BA D5 A9 F9 48 67 30 96 EE 94 78 5E 6F 89 A3 51 C0 30 86 66 A1 45 66 BA
54 EB A3 C3 91 F9 48 DC FF D1 E8 30 2D 7D 2D 74 70 35 D7 88 24 F7 9E C4 59 6E BB 73 87 17 F2 32
46 28 B8 43 FA B7 1D AA CA B4 F2 9F 24 0E 2D 4B F7 71 5C 5E 69 FF EA 95 02 CB 38 8A AE 50 38 6F
DB FB 2D 62 1B C5 C7 1E 54 E1 77 E0 67 C8 0F 9C 87 23 D6 3F 40 20 7F 20 80 C4 80 4C 3E 3B 24 26
8E 04 AE 6C 9A C8 AA 0D 02 03 01 00 01 A3 82 01 31 30 82 01 2D 30 0F 06 03 55 1D 13 01 01 FF 04
05 30 03 01 01 FF 30 0E 06 03 55 1D 0F 01 01 FF 04 04 03 02 01 86 30 1D 06 03 55 1D 0E 04 16 04
14 84 18 CC 85 34 EC BC 0C 94 94 2E 08 59 9C C7 B2 10 4E 0A 08 30 1F 06 03 55 1D 23 04 18 30 16
80 14 9C 5F 00 DF AA 01 D7 30 2B 38 88 A2 B8 6D 4A 9C F2 11 91 83 30 78 06 08 2B 06 01 05 05 07
01 01 04 6C 30 6A 30 2E 06 08 2B 06 01 05 05 07 30 01 86 22 68 74 74 70 3A 2F 2F 6F 63 73 70 2E
72 6F 6F 74 67 32 2E 61 6D 61 7A 6F 6E 74 72 75 73 74 2E 63 6F 6D 30 38 06 08 2B 06 01 05 05 07
30 02 86 2C 68 74 74 70 3A 2F 2F 63 72 74 2E 72 6F 6F 74 67 32 2E 61 6D 61 7A 6F 6E 74 72 75 73
74 2E 63 6F 6D 2F 72 6F 6F 74 67 32 2E 63 65 72 30 3D 06 03 55 1D 1F 04 36 30 34 30 32 A0 30 A0
2E 86 2C 68 74 74 70 3A 2F 2F 63 72 6C 2E 72 6F 6F 74 67 32 2E 61 6D 61 7A 6F 6E 74 72 75 73 74
2E 63 6F 6D 2F 72 6F 6F 74 67 32 2E 63 72 6C 30 11 06 03 55 1D 20 04 0A 30 08 30 06 06 04 55 1D
20 00 30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00 03 82 01 01 00 62 37 42 5C BC 10 B5 3E 8B 2C
E9 0C 9B 6C 45 E2 07 00 7A F9 C5 58 0B B9 08 8C 3E ED B3 25 3C B5 6F 50 E4 CD 35 6A A7 93 34 96
32 21 A9 48 44 AB 9C ED 3D B4 AA 73 6D E4 7F 16 80 89 6C CF 28 03 18 83 47 79 A3 10 7E 30 5B AC
3B B0 60 E0 77 D4 08 A6 E1 1D 7C 5E C0 BB F9 9A 7B 22 9D A7 00 09 7E AC 46 17 83 DC 9C 26 57 99
30 39 62 96 8F ED DA DE AA C5 CC 1B 3E CA 43 68 6C 57 16 BC D5 0E 20 2E FE FF C2 6A 5D 2E A0 4A
6D 14 58 87 94 E6 39 31 5F 7C 73 CB 90 88 6A 84 11 96 27 A6 ED D9 81 46 A6 7E A3 72 00 0A 52 3E
83 88 07 63 77 89 69 17 0F 39 85 D2 AB 08 45 4D D0 51 3A FD 5D 5D 37 64 4C 7E 30 B2 55 24 42 9D
36 B0 5D 9C 17 81 61 F1 CA F9 10 02 24 AB EB 0D 74 91 8D 7B 45 29 50 39 88 B2 A6 89 35 25 1E 14
6A 47 23 31 2F 5C 9A FA AD 9A 0E 62 51 A4 2A A9 C4 F9 34 9D 21 18
```


וכך למעשה התעודה מעוברת כאשר עולה הצורך להזדהות באמצעותה. המכונה בצד השני (ה-client לרוב) יקבל את רצף התווים (בצורה בינארית) ויבדוק שהם אכן תואמים למה ששמור ומוגדר אצלו לוקאלית.

לאחר שנחשפנו ליופי הנשגב של התעודה בפורמט ASN.1 ובתצורתה הבינארית לפי DER, ננתח את השדות (תכונות) שיש לנו בתעודת X.509 סטנדרטית.

זה פחות או מה שרובנו צריכים לדעת על המבנה של התעודה בכללי. במס' העמודים הקרובים אני אתעמק בערכים של השדות עצמם ומה ערך כזה או אחר אומר על תוכן התעודה. אם אתם לא נדרשים לעבוד בצורה ישירה עם תעודות, רפרוף קל שבקלים על התוכן של העמודים הבאים יספיק על מנת לתפוס את הרעיון הכללי לפני שניכנס לנושא של TLS. מצד שני, אם אתם אכן נדרשים לעבוד עם תעודות, אז ראשית כל - תנחומי. שנית כל, העמודים הבאים הם במיוחד בשבילכם.

ראשית, חשוב להכיר כי בדרך כלל נפגוש **10** שדות חובה אשר מצויים בכל תעודה (החל מגרסה 1) ומספר שדות אופציונאליים (אשר מומשו החל מגרסה 2 ו-3).

שדות ב-X.509v1

Version

מציין את מספר הגרסה של התעודה המקודדת. נכון לעכשיו, הערכים האפשריים של שדה זה הם 0, 1 או 2, אך יתכן ויהיה מורחב בעתיד. עובדה מעניינת היא שאף סטנדרט לא אוסר או מגביל לסמן תעודה v1 כתעודה v3 או להפך אבל, מן הראוי שיש להקפיד על זה. ערך השדה בפורמט ASN.1:

```
-----------------------------------------------------------------------
-- Version number
-----------------------------------------------------------------------
CertificateVersion ::= INTEGER {v1(0), v2(1), v3(2)}
```

Serial Number

מספר ייחודי שהוקצה לכל תעודה שהונפקה על ידי ה-CA (בדרך כלל, CA יחזיק counter בקובץ קבוע, config file תחת Unix ורגיסטרי ב-windows). דרך נוספת למימוש היא לחסר את הזמן הנוכחי (בשניות) מקבוע זמן כלשהו כגון זמן הרצת התוכנה הראשוני. שיטה זו עדיפה על השיטה האינקרמנטלית מכיוון שהיא אינה מאפשרת מעקב אחרי רצף התעודות וחשיפת מידע רגיש (למשל מספר התעודות ש CA הפיק בשבוע עבודה).

דוגמא ל-Serial Number של תעודה: f944a2a27cdf3fac2ae2b01f908eeb9c4c6067

```
-----------------------------------------------------------------------
-- Certificate serial number
-----------------------------------------------------------------------
CertificateSerialNumber ::= INTEGER
```

Signature Algorithm

מכיל OID (object identifier) שמציין את האלגוריתם שה-CA חתם איתו את התעודה. למשל, 2.840.113549.1.1.51. מציין את גיבוב SHA-1 עם הצפנת RSA. לא נראה שיש שימוש רב לשדה הנ"ל (מכיוון אפשר לבדוק את אלגוריתם החתימה גם בחתימה עצמה שעל התעודה. קיים ספק אם מדובר באמצעי הגנה בגלל שבמידה ולמישהו הייתה גישה לחתימה אז גישה לאלגוריתם החתימה הוא פועל יוצא מכך).

רלוונטי לציין שיש להיות מאוד זהירים עם השימוש ב-OID. ברוב המקרים יש מספר OIDs לאותו אלגוריתם אך יש צורך לבחור OID ספציפי.

נקודה נוספת שיש לשים לב אליה היא לא להשאיר את שדה הפרמטרים ריק במידה ולא קיימים משתנים אלא להשתמש ב -NULL.

דוגמא ל-Signature algorithm: sha256RSA:

```
----------------------------------------------------------------------
-- Signature OID
----------------------------------------------------------------------
signature ::= AlgorithmIdentifier

AlgorithmIdentifier ::= SEQUENCE
{
  algorithm            OBJECT IDENTIFIER,
  parameters           ANY OPTIONAL
}
```

Issuer

מכיל את DN (Distinguished name) של ה-CA שיצר וחתם על התעודה. DN הוא ה-path דרך עץ X.500 directory.

דוגמא לתוכן השדה Issuer בתעודה של amazon.com:

```
CN = Starfield Services Root Certificate Authority - G2
O = Starfield Technologies, Inc.
L = Scottsdale
S = Arizona
C = US
```

```
----------------------------------------------------------------------
-- Issuer name
----------------------------------------------------------------------
Name ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET OF AttributeTypeValue

AttributeTypeValue ::= SEQUENCE
{
  type        OBJECT IDENTIFIER,
  value       ANY
}
```

Validity

מציין את מרווח הזמן שבמהלכו התעודה תקפה (yymmddhhmmssz). התאריך הראשון הוא הזמן בו התעודה נכנסה לתוקף והתאריך השני הוא התאריך בה התעודה בטלה.

נושא שחשוב להבין הוא שבדרך כלל CA מטפל בסוגייה של תעודות שעבר זמנן (או שפקיעת זמנן קרב) על ידי כך שהוא פשוט מפרסם תעודות חדשות עם אותו שם ומפתח אבל עם שדה Validity שונה. במקרים מסוימים, אפילו CA roots פורסמו מחדש רק עם שדה הנ"ל שונה. שיטה כזו יכולה להוביל למצב בו יש כמה תעודות "זהות" שבתוקף בו-בזמן.

סמנטיקה של תעודות / מפתחות מהסוג הזה לא ברורה. אולי פשוט ניתן לשנות את שם השדה ל-*RenewalFeeDueDate*. בהתחשב בעובדה שקיימת אפשרות להנפיק תעודות עם תוקף של עשר שנים וצפונה ועדיין CA-ים להנפיק רק לשנה-שנתיים, אופציה זו לא מבוטלת.

```
---------------------------------------------------------------------
-- Validity period
---------------------------------------------------------------------
Validity ::= SEQUENCE
{
  notBefore           ChoiceOfTime,
  notAfter            ChoiceOfTime
}

ChoiceOfTime ::= CHOICE
{
  utcTime                 UTCTime,
  generalTime             GeneralizedTime
}
```

Subject

השדה מכיל את ה-X.500 DN של הישות שמחזיקה במפתח הפרטי אשר קשור למפתח הציבורי שנמצא בתעודה. תוכן השדה יכול לתאר את CA, RA או EE.

שדות ב-**X.500 DN** כפי שמוכלות בתעודות X.509 מכילים את הפרמטרים הבאים:

- **CN**:CommonName - שם של אדם. "Susan Jones"
- **OU**:OrganizationUnit - מחלקה בארגון. "Purchasing"
- **O**:OrganizationName - שם הארגון. "Amazon"
- **L**:LocalityName - שם עיר. "Palo Alto"
- **S**:StateName - שם מדינה. "California"
- **C**:Country - 2 אותיות לקוד המדינה. "CH"

דוגמא לתוכן השדה Subject בתעודה של amazon.com:

```
CN = Amazon Root CA 1
O = Amazon
C = US
```

```
---------------------------------------------------------------------
-- Subject name
---------------------------------------------------------------------
Name ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET OF AttributeTypeValue

AttributeTypeValue ::= SEQUENCE
{
  type        OBJECT IDENTIFIER,
  value       ANY
}
```

Public Key

שדה זה מכיל את המפתח הפרטי והאלגוריתם המזוהה עם המפתח (כדוגמת RSA, DSA, Diffie-Hellman).

```
-----------------------------------------------------------------------
--  Subject public key information
-----------------------------------------------------------------------
SubjectPublicKeyInfo ::= SEQUENCE
{
  algorithm            AlgorithmIdentifier,
  subjectPublicKey     BITSTRING
}

AlgorithmIdentifier ::= SEQUENCE
{
  algorithm            OBJECT IDENTIFIER,
  parameters           ANY OPTIONAL
}
```

שדות ב-X.509v2

2 שדות אופציונאליים נוספו ל-X.509 בגרסה 2 כחלק מהסטנדרט. מטרת השדות כשמם כן היא - לזהות מצב תיאורטי בו נעשה שימוש חוזר בשם ה-Issuer ו\או Subject. רוב הדוקומנטציה בנושא ממליצה לא למחזר שמות (ולא להשתמש בשדות של Unique Identifier), כמו כן, ארגון IETF החליט לקטוע את השימוש בהם אז למעשה **אין צורך לממש אותם בתעודות.** עם זאת, מכיוון שהן חלק מהסטנדרט, במידה ורושמים קוד שמתעסק בתעודות צריך לקחת את השדות האלה בחשבון.

Issuer Unique Identifier

bitstring שמזהה Issuer X.500 name במקרה ונקבע שם דומה ל-CA בעבר.

```
-----------------------------------------------------------------------
-- Issuer Unique identifier
-----------------------------------------------------------------------
issuerUniqueIdentifier ::= [1] IMPLICIT UniqueIdentifier OPTIONAL

UniqueIdentifier ::= BITSTRING
```

Subject Unique Identifier

למקרה וצצים RA, CA, EE עם שמות שכבר קיימים. יש הסבורים כי שדה זה מיותר מכיוון שאפשר לחסות את האופציות שחופפות באמצעות השדה של SerialNumber ב-X.500.

```
-----------------------------------------------------------------------
-- Issuer Unique identifier
-----------------------------------------------------------------------
subjectUniqueIdentifier ::= [2] IMPLICIT UniqueIdentifier OPTIONAL

UniqueIdentifier ::= BITSTRING
```

שדות ב-X.509v3

טוב טוב, פה למעשה יש הכי הרבה "בשר" (או 'טופו' אם אתם שומרים, הכל בסדר זה רק פיקסלים על מסך). יש המון יתרונות ופונקציונאליות לגרסה 3 של התעודות. כמובן שעם פונקציונאליות רבה מגיעות פגיעויות.

גרסה 3 של X.509 מציגה מנגנון לפיו ניתן "להרחיב" תעודות בצורה סטנדרטית וגנרית על מנת לכלול מידע נוסף בתעודות עצמן. ניתן להוסיף מספר הרחבות לתעודה. ההרחבות מספקות מידע הקשור *לשימוש במפתח, מדיניות שימוש בתעודה, הגבלות על מבנה שרשור התעודות, הפצת CRL, הרחבות פרטיות, תצורות שם וכו'.* כבר בשלב זה חשוב לציין - **קיים מספר רב של הרחבות** ([RFC-5280](RFC-5280) מדבר על 16 וב-oidref.com יש 69 סה"כ), לא כולן בשימוש אקטיבי ו\או בעלי חשיבות מרובה לכל מערכת באשר היא.

מונחים כגון 'תוספי תקן' (**standard extensions**) מתייחסים לעובדה שהתקן בגרסה 3 מגדיר תוספות שכיחות לתעודה שלרוב נעשה בהן שימוש. עם זאת, תעודות אינן מוגבלות רק לתוספים הסטנדרטיים וכל אחד יכול להוסיף הרחבה ברשויות המתאימות (כגון ISO).

כל הרחבה מורכבת מ-3 שדות: Type, Criticality, Value. תוכן ההרחבות בפורמט ASN.1:

```
--------------------------------------------------------------------
-- Extensions (beginning with version 3).
--------------------------------------------------------------------
Extensions ::= SEQUENCE OF Extension

Extension ::= SEQUENCE
{
  Id                    OBJECT IDENTIFIER,
  critical              BOOLEAN DEFAULT FALSE,
  extnValue             OCTET STRING
}
```

כפי שניתן לראות, בכל הרחבה קיים **ערך OID**, ערך בוליאני המסמן האם ההרחבה מסווגת כ**קריטית** ושדה byte array שמכיל את הערך של ההרחבה.

יש הבחנה חשובה בין הרחבה **קריטית** להרחבה "רגילה" של מידע בתעודה. יישום מסוים עשוי לדרוש שתוסף ספציפי יהיה זמין בכל תעודה שעובדת ביישום עצמו. אבל עם זאת, זה לא אומר כי הצורך כי לסווג את ההרחבה כקריטית. הרחבות קריטיות מיועדות רק למידע חשוב מאוד - כגון מידע קריטי למניעת שימוש לרעה בתעודה או שימוש לא בטוח בתעודה. כתוצאה מכך, הרוב המכריע של ההרחבות אינן קריטיות. יש לסווג הרחבה כקריטית רק לאחר הבנה מעמיקה והתחשבות בכך שזה יכול ליצור בעיות הדדיות עם יישומי תחומי אבטחה אחרים של אותו CA.

במידה ולא ניתן לזהות ו\או לאכוף שדה של הרחבה קריטית חובה **לדחות את כל התעודה** על הסף.

רק לשם הבנת גודל הסוגייה שעומדת לפנינו בכל הנוגע להרחבות ולשימוש בהן - **חצי מאורך התקן** שמייצג את כל נושא תעודות X.509 **עוסק אך ורק ב-X.509v3 extensions**. עם זאת, מכיוון שהתקן לא מגדיר באופן מפורש כיצד יש לעבד את המידע שמסופק בהן, גופי CA, RA, EE בוחרים לממש, לבדוק ולאכוף בצורה שונה - **מה שגורר בעיות רבות.**

בכל מקרה, בין אם תבחרו לקרוא את הפרק הבא או בין אם לא, חשוב לי מאוד שתקראו ותבינו את המשפט הבא מכיון שהוא המשפט הכי קריטי בכל הנוגע X.509 extensions ולעבודה עימן בתעודות.

ההרחבות basicConstraints, exKeyUsage, KeyUsage פועלות ביחד למתן הגדרה כיצד יש להשתמש בתעודה. יישומים יכולים להשתמש בהרחבות אלו כדי למנוע שימוש לא מורשה בתוכן התעודה.

פירוט מעמיק על X.509v3 extensions

מישהו ביקש עוד drill down על השדות בתעודה ולא קיבל? תת פרק זה מפרט בהרחבה על סוגי ההרחבות (extensions) המוגדרים כחלק מתקן X.509 גרסה 3 ומציין אילו סוגים מומלצים וכיצד יש לסווג את חשיבותם.

במסגרת המחקר הוחלט להרחיב רק על השדות extensions הבאים מכיוון שהם שכיחים ונתמכים על ידי מיקרוסופט. ישנם מס' לא מבוטל של הרחבות שלא נכללו במכון והמסמך לא מרחיב עליהם.

ראשית, ניזכר במבנה התעודה, כאשר נקדיש תשומת לב פרטנית ואוהבת לשדות ה-extensions. נריץ את הפקודה:

```
C:\Users\Jonathan Elkabas\Desktop>openssl x509 -in google.cer -text
```

ונקבל:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7e:10:d9:01:f7:ac:03:cd:08:00:00:00:00:47:ef:8e
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=Google Trust Services, CN=GTS CA 1O1
        Validity
            Not Before: Jun 17 14:23:06 2020 GMT
            Not After : Sep  9 14:23:06 2020 GMT
        Subject: C=US, ST=California, L=Mountain View, O=Google LLC, CN=*.google.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:e2:0f:c2:89:99:94:1d:58:ce:db:11:2f:1c:d2:
                    12:d5:47:7a:81:16:3f:df:86:65:ae:d3:43:ba:9e:
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
```

```
X509v3 Basic Constraints: critical
    CA:FALSE
X509v3 Subject Key Identifier:
    33:3F:50:E1:30:43:87:FB:F4:35:B6:E0:5D:6B:50:D9:BA:81:C8:BD
X509v3 Authority Key Identifier:
    keyid:98:D1:F8:6E:10:EB:CF:9B:EC:60:9F:18:90:1B:A0:EB:7D:09:FD:2B

Authority Information Access:
    OCSP - URI:http://ocsp.pki.goog/gts1o1core
    CA Issuers - URI:http://pki.goog/gsr2/GTS1O1.crt

X509v3 Subject Alternative Name:
    DNS:*.google.com, DNS:*.android.com, DNS:youtube.com, DNS:youtubeeducation.com
X509v3 Certificate Policies:
    Policy: 2.23.140.1.2.2
    Policy: 1.3.6.1.4.1.11129.2.5.3

X509v3 CRL Distribution Points:
    Full Name:
    URI:http://crl.pki.goog/GTS1O1core.crl

Signature Algorithm: sha256WithRSAEncryption
    6d:a6:b4:f6:4a:63:bf:ed:80:20:b6:eb:01:85:2e:cb:41:fa:
    10:01:cf:a6:bf:eb:20:99:2c:59:2e:6d:f2:8d:c6:5f:ac:03:
    ................
```

לאחר שראינו איך נראים השדות בתעודה עצמה, נעמיק על מס' שדות extension שכיחים (מצורף ייצוג של כל הרחבה בפורמט ASN.1 לשם הבנה כללית של תוכן השדות):

authorityKeyIdentifier

```
AuthorityKeyIdentifier ::= SEQUENCE {
   keyIdentifier             [0] KeyIdentifier           OPTIONAL,
   authorityCertIssuer       [1] GeneralNames            OPTIONAL,
   authorityCertSerialNumber [2] CertificateSerialNumber OPTIONAL   }

KeyIdentifier ::= OCTET STRIN
```

הרחבת Authority Key Identifier מסווג את המפתח הציבורי המתאים למפתח הפרטי המשמש לחתימת התעודה. הרחבה זו שימושית כאשר המנפיק עושה שימוש במספר מפתחות חתימה, כגון כאשר תעודת CA עוברת חידוש.

ההרחבה מורכבת מאחד או משני השדות הפנימיים הבאים:

1. מזהה מפתח מפורש, אשר מוגדר בשדה KeyIdentifier.
2. מנפיק, אשר מוגדר בשדה authorityCertIssuer ומספר סריאלי אשר מזהה את התעודה, בשדה authorityCertSerialNumber.

- אם שדה KeyIdentifier קיים, משתמשים בו על מנת לבחור בתעודה עם extension של subjectKeyIdentifier מתאים.

- אם authorityCertIssuer ו-authorityCertSerialNumber קיימים אז עושים בהם שימוש לזיהוי התעודה המתאימה לפי ה-issuer וה-serialNumber.

אם הרחבה זו לא קיימת בתעודה עושים שימוש ב-issuer name **בלבד** על מנת לוודא את מנפיק התעודה.

OID: 5.29.352. **Criticality:** הרחבה זו תמיד תהיה מוגדרת כ-noncritical ותמיד יש לבדוק אותה.

basicConstraints

```
BasicConstraints ::= SEQUENCE {
        cA                      BOOLEAN DEFAULT FALSE,
        pathLenConstraint       INTEGER (0..MAX) OPTIONAL }
```

נעשה שימוש בהרחבה זו תוך כדי תהליך אימות ה-certificate chain לזיהוי תעודת CA ולאכיפה של הגבלת מספר שרשור התעודות. שדה cA צריך להיות true לכל תעודות CA. מומלץ לא להכיל שדה זה בתעודות EE.

- אם שדה pathLenConstraint קיים (שדה זה מגביל את מספר התעודות שמתאפשר בשרשור נתון), ערכו חייב להיות גדול יותר ממספר תעודות CA שעוקבו עד כה, החל מתעודות EE והתקדמות במעלה השרשור.
- אם pathLenConstraint לא קיים אז אין מגבלה על מספר התעודות בשרשור ועל כל תעודות CA במעלה השרשרת לא להכיל את הערך הזה גם כן.

OID: 2.5.29.19. **Criticality:** הרחבה זו תמיד מוגדרת כ-**critical** ותמיד יש לבדוק אותה ללא קשר להיות קריטית או לא.

certificatePoliciesExt

```
certificatePolicies ::= SEQUENCE SIZE (1..MAX) OF PolicyInformation

PolicyInformation ::= SEQUENCE {
     policyIdentifier   CertPolicyId,
     policyQualifiers   SEQUENCE SIZE (1..MAX) OF
                             PolicyQualifierInfo OPTIONAL }

CertPolicyId ::= OBJECT IDENTIFIER

PolicyQualifierInfo ::= SEQUENCE {
     policyQualifierId  PolicyQualifierId,
     qualifier          ANY DEFINED BY policyQualifierId }
```

הרחבת Certificate Policies מגדירה אחת או יותר מדיניות, כל אחד מורכבת מ-OID ומספר אופציונאלי של qualifiers. הרחבה יכולה לכלול URI (Uniform Resource Identifier, לדוגמא wiki.org) למדריך שימוש של המנפיק או לקטע טקסט על המנפיק עצמו. במידה והרחבה זו קיימת, מומלץ שהמדיניות יוגדרו באמצעות OID בלבד.

OID: 5.29.322. **Criticality:** critical or noncritical

CRLDistributionPoints

```
CRLDistributionPoints ::= SEQUENCE SIZE (1..MAX) OF DistributionPoint

DistributionPoint ::= SEQUENCE {
     distributionPoint        [0]     DistributionPointName OPTIONAL,
     reasons                  [1]     ReasonFlags OPTIONAL,
     cRLIssuer                [2]     GeneralNames OPTIONAL }

DistributionPointName ::= CHOICE {
     fullName                 [0]     GeneralNames,
     nameRelativeToCRLIssuer  [1]     RelativeDistinguishedName }
```

הרחבה זו מגדירה כיצד מידע CRL-מ מתקבל. יש לעשות שימוש בשדה זה רק במידה והיישום\ מערכת עושה שימוש ב-CRL issuing points.

- אם ההרחבה כוללת את השדה DistributionPointName עם type של URI, ניתן להניח שה-URI מצביע ל-CRL הנוכחי למתן סיבות פסילת התעודות ויונפק על ידי השם ב-cRLIssuer. הערכים המצופים ל-URI הם אלו שמוגדרים בהרחבה subjectAltName.
- אם שדה distributionPoint לא כולל סיבות בכלל, ה-CRL חייב לכלול את כל סיבות לפסילה בעצמו.
- אם שדה distributionPoint מחסיר את שדה cRLIssuer ה-CRL חייב להיות מונפק על ידי ה-CA שהנפיק את התעודה.

מומלץ כי CAs ואפליקציות יעשו שימוש בהרחבה זו.

OID: 5.29.312. **Criticality:** ההרחבה מוגדרת כ-noncritical ומומלצת לתמוך בכלל התעודות.

keyUsage

```
KeyUsage ::= BIT STRING {
     digitalSignature        (0),
     nonRepudiation          (1), -- recent editions of X.509 have
                           -- renamed this bit to contentCommitment
     keyEncipherment         (2),
     dataEncipherment        (3),
     keyAgreement            (4),
     keyCertSign             (5),
     cRLSign                 (6),
     encipherOnly            (7),
     decipherOnly            (8) }
```

הרחבת KeyUsage מגדירה את היעוד לשימוש במפתח הציבורי שנמצא בתעודה. במידה וההרחבה מוגדרת כ-critical, היא אוכפת את השימוש בתעודה ובמפתח. אם ההרחבה לא קיימת\ לא מוגדרת כקריטית בתעודה, כל סוגי השימוש מותרים.

בתוך ההרחבה יש 9 ביטים אשר מציינים את השדות (flags) הבאים:

- **(0) digitalSignature** - מקבל את הערך 1 כאשר המפתח הציבורי של בעל התעודה משמש לאימות חתימות דיגיטליות (מלבד אלו שעל התעודות עצמן). בדרך כלל נראה שימוש בו לתעודות SSL client, תעודות S/MIME ולתעודות המשמשות לחתימה על אובייקט עצמאי.
- **(1) nonRepudiation** - פעולה דומה לשדה digitalSignature אך העלת סיביב של שדה זה לערך 1 מחייב את הישות שחתמה על התעודה לתת שירות. השימוש בסיבית זו שנוי במחלוקת ויש לשקול בזהירות את ההשלכות החוקיות של השימוש בה לפני שמקצים אותה לתעודה.
- **(2) keyEncipherment** - כאשר המפתח הציבורי שבתעודה משמש להצפנת מפתח פרטי\ סודי. (key transport). למשל, כאשר עושים שימוש במפתח RSA ציבורי להצפנת מפתח פרטי סימטרי.
- **(3) dataEncipherment** - כאשר המפתח הציבורי משמש ישירות להצפנת raw data של המשתמש. מאוד לא נפוץ.
- **(4) keyAgreement** - כאשר המפתח הציבורי משמש בתור key agreement. למשל כאשר עושים שימוש במפתח Diffie-Helman ל-key management.
- **(5) keyCertSign** - כאשר עושים שימוש במפתח הציבורי לאימות החתימה של מפתח ציבורי בתעודה. כאשר מעלים סיבית זו ל-1 חובה לעלות גם את סיבית cA בהרחבת basicConstraints. במילים פשוטות, כל CA שחותם תעודות יעשה בזה שימוש.
- **(6) cRLSign** - כאשר נעשה שימוש במפתח הציבורי לאימות חתימות דיגיטליות ב-CRLים.
- **(7) encipherOnly** - כאשר המפתח הציבורי משמש רק להצפנת נתונים בזמן ביצוע key agreement. אם סיבית זו מקבלת את הערך 1, גם keyAgreement חייבת.
- **(8) decipherOnly** - כאשר המפתח הציבורי משמש רק לפענוח נתונים בזמן ביצוע key agreement. אם סיבית זו מקבלת את הערך 1, גם keyAgreement חייבת.

כאשר הרחבת KeyUsage קיימת בתעודה **אסור** להשתמש בשדה subject public key לאימות חתימות על תעודות או על CRLים אלא אם סיביות KeyCertSign או cRLSign בערך 1.

OID: 5.29.152. **Criticality:** הרחבה זו יכולה לסווג כ-critical או noncritical. מומלץ לסמנה כקריטית במידה ועושים בה שימוש.

extKeyUsage

```
ExtKeyUsageSyntax ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId

KeyPurposeId ::= OBJECT IDENTIFIER
```

הרחבת Extended Key Usage מגדירה את היעודים לשימוש במפתח הציבורי של התעודה. היעודים הללו יכולים להיות בנוסף או במקום ליעודים שהוגדרו בהרחבת KeyUsage.

טבלה המפרטת את השימוש להרחבה זו:

Use	OID
Server authentication	**1.3.6.1.5.5.7.3.1**
Client authentication	**1.3.6.1.5.5.7.3.2**
Code signing	**1.3.6.1.5.5.7.3.3**
Email	**1.3.6.1.5.5.7.3.4**
Timestamping	**1.3.6.1.5.5.7.3.8**
OCSP Signing	**1.3.6.1.5.5.7.3.9**

OID: 5.29.372. **Criticality:** אם הרחבה זו מסומנת כ-critical, יש להשתמש בתעודה לאחת המטרות שצוינו בלבד. אם היא מסומנת כ-noncritical, השדה מתייחס הוא לייעוץ בלבד ומשמש לזיהוי מפתחות אך אינו מגביל את השימוש בתעודה למטרות שצוינו.

issuerAltName

```
IssuerAltName ::= GeneralNames
```

הרחבת Issuer Alternative Name משמשת לקישור זהויות בסגנון אינטרנטי עם ה-CA. שמות חייבים להשתמש בתצורה שהוגדרה בהרחבה של subjectAltName.

OID: 5.29.182. **Criticality:** noncritical

nameConstraints

```
NameConstraints ::= SEQUENCE {

     permittedSubtrees        [0]     GeneralSubtrees OPTIONAL,
     excludedSubtrees         [1]     GeneralSubtrees OPTIONAL }
```

ההרחבה מגדירה מרחב שמות (name space) אשר בו נכללים כל subject names לתעודות הבאות בתוך certification path. ההגבלות חלות ל-subject distinguished name ול-subject alternative names. למשל, ל-URIs, ה-name constrains קשורות לחלק של ה-host בשם. רק תעודות CA ממשים הרחבה זו.

OID: 5.29.302. **Criticality:** מומלץ להיות critical.

policyMappings

```
PolicyMappings ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
        issuerDomainPolicy      CertPolicyId,
        subjectDomainPolicy     CertPolicyId }
```

Policy mappings מגדיר את רשימת המדיניות שקשורים ל-CA מסוים העשוי להתקבל כמשווה למדיניות של המנפיק, כלומר מה חל על בעל התעודה שחל גם על ה-CA השני. שימושי כאשר ממומש cross-pair certificates.

ההרחבה מספקת אחד או יותר צמדים של OIDs, כאשר בכל צמד יש שני שדות - issuerDomainPolicy, subjectDomainPolicy אשר מתארים את טיב היחסים מבחינת מדיניות בין 2 CAs. רק תעודות CA ממשים הרחבה זו.

OID: 5.29.332. **Criticality:** noncritical

policyConstraints

```
PolicyConstraints ::= SEQUENCE {
        requireExplicitPolicy           [0] SkipCerts OPTIONAL,
        inhibitPolicyMapping            [1] SkipCerts OPTIONAL }

   SkipCerts ::= INTEGER (0..MAX)
```

ההרחבה משמשת לתעודות CA בלבד. היא משמשת למתן מגבלות על path validation בשתי דרכים:

- איסור שימוש ב-policy mapping
- חיוב שימוש ב-policy identifier בכל תעודה שנמצאת ב-path.

אם שדה inhibitPolicyMapping קיים בתעודה, הערך שלו מציין את מספר התעודות הנוספות שיכולות להיות ב-path לפני ש-policy mapping לא מאופשר יותר.

אם שדה inhibitPolicyMapping קיים בתעודה, הערך שמוצג בו מציין את מספר התעודות שיכולות להיות ב-path לפני שנדרש קיום של מדיניות מפורשת לכל ה-path.

OID: 5.29.362. **Criticality:** CAs מאמתים חייבים להגדיר את ההרחבה הזו כ-critical. מלבד זה, הרחבה יכולה לסווג noncritical או critical בהתאם לאופן השימוש.

subjectAltName

```
SubjectAltName ::= GeneralNames

   GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

   GeneralName ::= CHOICE {
        otherName                       [0]     OtherName,
        rfc822Name                      [1]     IA5String,
        dNSName                         [2]     IA5String,
        x400Address                     [3]     ORAddress,
        directoryName                   [4]     Name,
        ediPartyName                    [5]     EDIPartyName,
        uniformResourceIdentifier       [6]     IA5String,
        iPAddress                       [7]     OCTET STRING,
        registeredID                    [8]     OBJECT IDENTIFIER }
```

ההרחבה מאפשרת לקשור זהויות למחזיק במפתח הפרטי שקשור למפתח הציבורי שבתוך התעודה. זהויות אלו יכולות להיכלל בנוסף או במקום הזהות בשדה subject name. ההרחבה כוללת אחד או יותר שמות אלטרנטיביים (לא X.500 DN בהכרח) לזהות המחוברת על ידי ה-CA למפתח הציבורי שבתעודה. השמות הכלליים יכולים להיות כתובות מייל, שמות DNS, כתובות IP, URI וכו'.

כאשר קושרים בתעודה זהויות נוספות למפתח הציבורי, חייבים לעשות שימוש בהרחבה זו.

הערה: לא מוגדר בתקן שימוש בשדה זה כאשר נעשה שימוש ב-wildcard char (*).

OID: 5.29.172. **Criticality:** אם שדה subject name בתעודה ריק, שדה זה חייב לסווג כ-critical.

subjectDirectoryAttributes

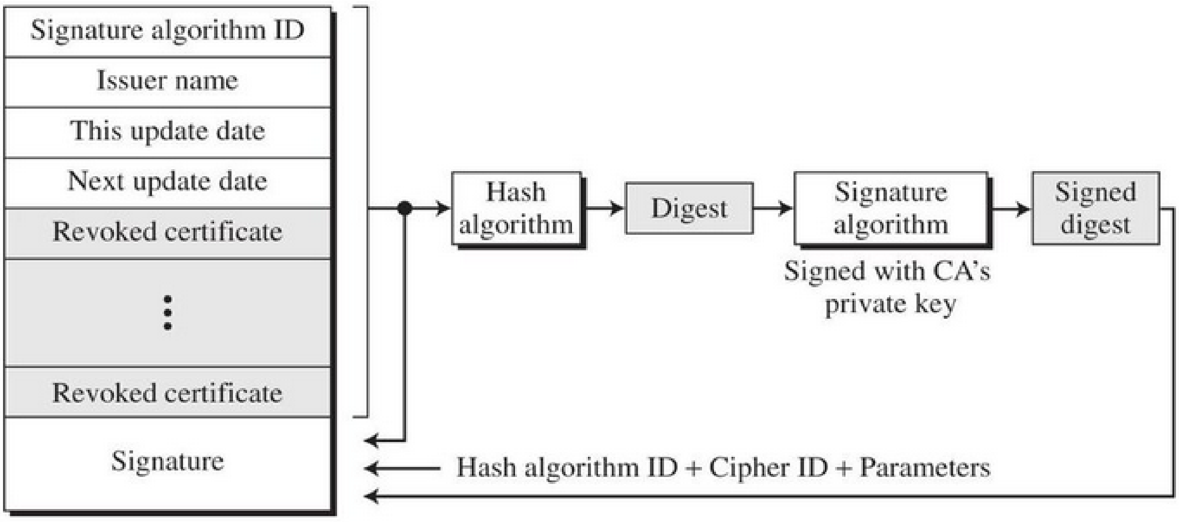
```
SubjectDirectoryAttributes ::= SEQUENCE SIZE (1..MAX) OF Attribut
```

הרחבה זו משמשת להעברת מאפייני attributes רצויים (כגון לאום ופרטים נוספים שלא הוגדרו בשדות הבסיסיים של התעודה) של בעל התעודה. ההרחבה מורכבת מרצף של אחד או יותר מאפיינים ועושים בה שימוש בעיקר בסביבות לוקאליות.

OID: 5.29.92. **Criticality:** noncritical

פורמט CRLv2

זוכרים שדיברנו על CRL? בשורה אחת - רשימה שאומרת אילו תעודות נגנזו טרם זמנן. מעולה, כי שדות של רשימות ביטול תעודות דומות בצורתן למבנה התעודה עצמה והן עוברות תהליך חתימה דומה:



פורמט ASN.1 של CRL:

```
CertificateList  ::=  SEQUENCE  {
        tbsCertList          TBSCertList,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING  }
```

כאשר באופן דומה לתעודות, TBSCertList מוגדר על ידי:

```
TBSCertList  ::=  SEQUENCE  {
       version                 Version OPTIONAL,
                                    -- if present, shall be v2
       signature               AlgorithmIdentifier,
       issuer                  Name,
       thisUpdate              Time,
       nextUpdate              Time OPTIONAL,
       revokedCertificates     SEQUENCE OF SEQUENCE  {
            userCertificate         CertificateSerialNumber,
            revocationDate          Time,
            crlEntryExtensions      Extensions OPTIONAL
                                          -- if present, shall be v2
                                 }  OPTIONAL,
       crlExtensions           [0]  EXPLICIT Extensions OPTIONAL
                                          -- if present, shall be v2
                                 }
```

שדות ב-CRL:

Version, Signature, Issuer Name, This Update, Next Update, Revoked Certificates, Extensions.

יש לשים לב כי גם CRLv2 מכיל שדה של extensions.

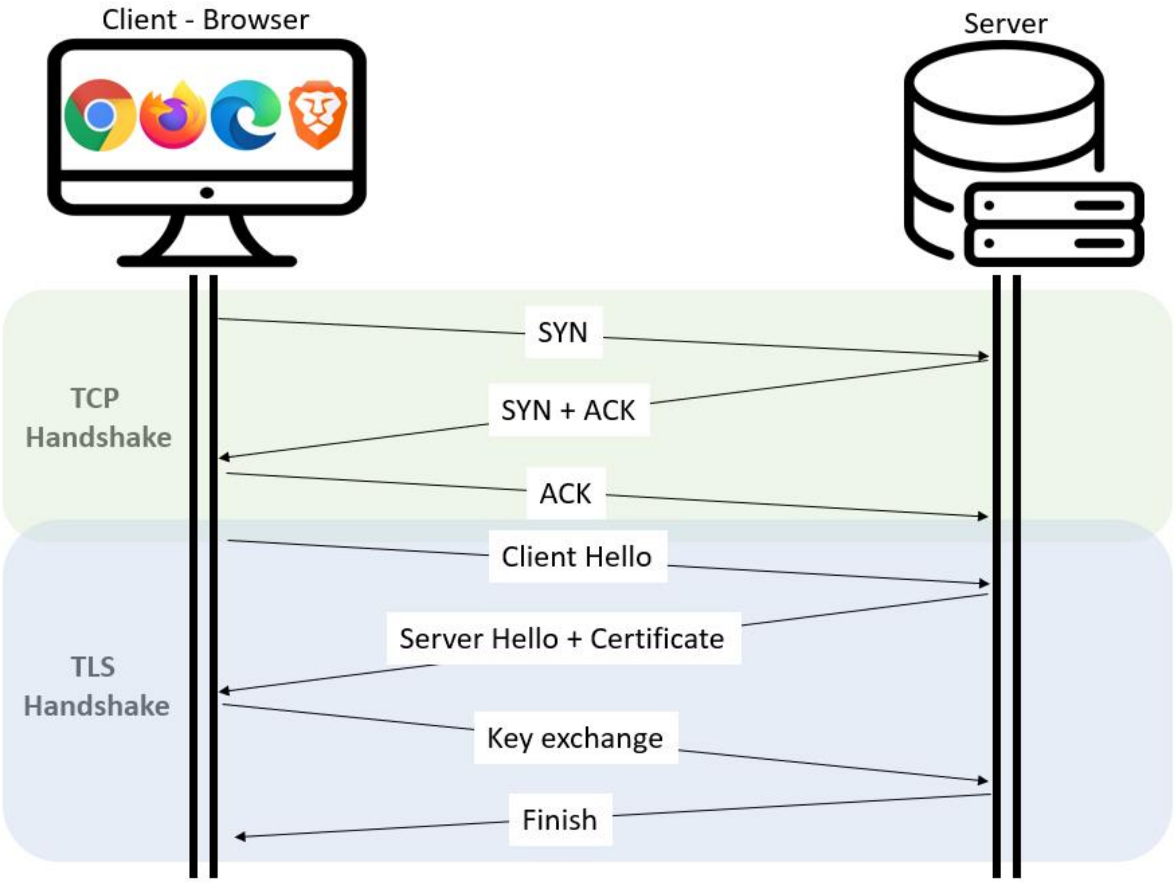
מכיוון שנושא המרכז של המסמך הינו תעודות דיגיטליות, אני לא מוצא לנכון להכביד במילים על אופן המימוש והעבודה של CRL-ים. תפעול והפצה של CRL-ים הוא נושא מורכב (וחשוב) מאוד ואפשר לרשום עליו מסמך באורך זהה.

ניתן לקרוא עוד על מימוש, הפצה, ניהול ותהליך יצירת CRL ב-RFC2459 (לא ממליץ בחום).

ללחוץ ידיים בביטחון - TLS Handshake

אז הגענו למסקנה ש-א' אנחנו רוצים תעודות בשביל לוודא שאנחנו מדברים עם מישהו מהימן ו-ב' אנחנו רוצים שהתעבורה בנינו תהיה מוצפנת כדי שנוכל לשלוח לו מידע רגיש בסבבה. אבל איך בדיוק אנחנו מעבירים ו\או מקבלים את אותן תעודות?

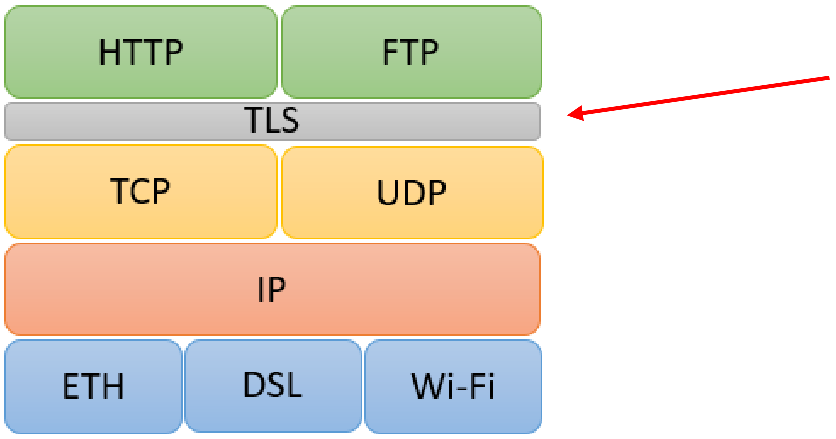
בואו נגיד שבא לי לקנות שטות אימפולסיבית מ-Amazon, מתי Amazon יבוא ויגיד לי "שומע אחי, אני אחלה אתה יכול לסמוך עליי ולשים פה את פרטי האשראי שלך"? וגם, איך בדיוק אנחנו אמורים להצפין את המידע בנינו?



תקשורת HTTP מתבצעת על גבי TCP connection. חיבור TCP הוא מוכוון חיבור, מה הכוונה? כל פרוטוקול מוכוון חיבור צריך ליצור חיבור על מנת לשריין משאבים בשני הקצוות המתקשרים. הסכמה על TCP connection נעשית בתהליך תלת שלבי שנקרא TCP handshake. אני מניח שכל זה מוכר.

TLS מהצד שלו, לוקח את תהליך חיבור ה-TCP ומוסיף עליו את ה-headers שלו - עוד 4 לחיצות ידיים. הם מאפשרים 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.

אם נסתכל על שילוב הפרוטוקול בממשקים שלו עם פרוטוקולים אחרים נראה ש-TLS רץ מעל TCP (שכבת התעבורה) ומתחת ל- HTTP ופרוטוקולים אחרים ברמת האפליקציה. כלומר, הוא לא משנה את אופן העבודה של HTTP:



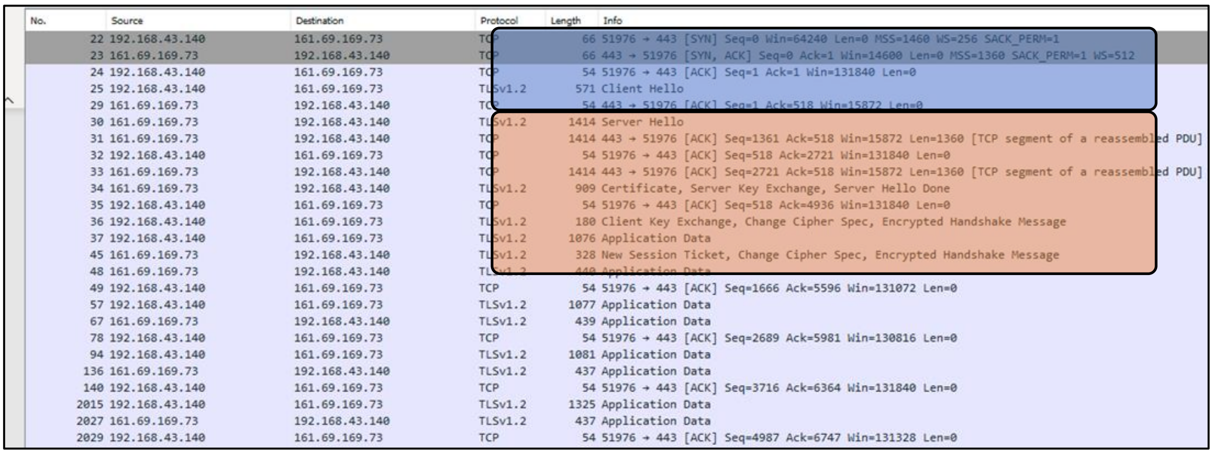
כפי שנראה עוד מעט ב-Wireshark, TLS מבצע אנקפסולציה (כימוס בעברית צחה) לשכבות שמעליו וככה שולח את המידע שמגיע מהן.

חשוב לי לעצור שנייה ולתת קרדיט כחול לבן ענק למאמר SSL & Trasport Layer Security Protocol של **אפיק קסטיאל** מגליון 2 (עוד ב-2009..!) ולמאמר TLS חלק א' (הצפנות א-סימטריות, RSA הבסיס המתמטי) של **שחר קורוט** מגליון 85 של DW ולמאמר HTTPS למתקדמים: אימות זהות של ליאור-בר און.

למדתי מהם המון על TLS והם עזרו לי רבות בכתיבת הפרק! ממליץ לכולם לקרוא את המאמרים שהם פרסמו, קיים שם פירוט תיאורטי ומתמטי **מעולה**.

תקציר תהליך ה TLS handshake:

על מנת לספק דוגמא כללית לתהליך, נתחבר ל-amazon.com ונסניף את התעבורה באמצעות Wireshark.

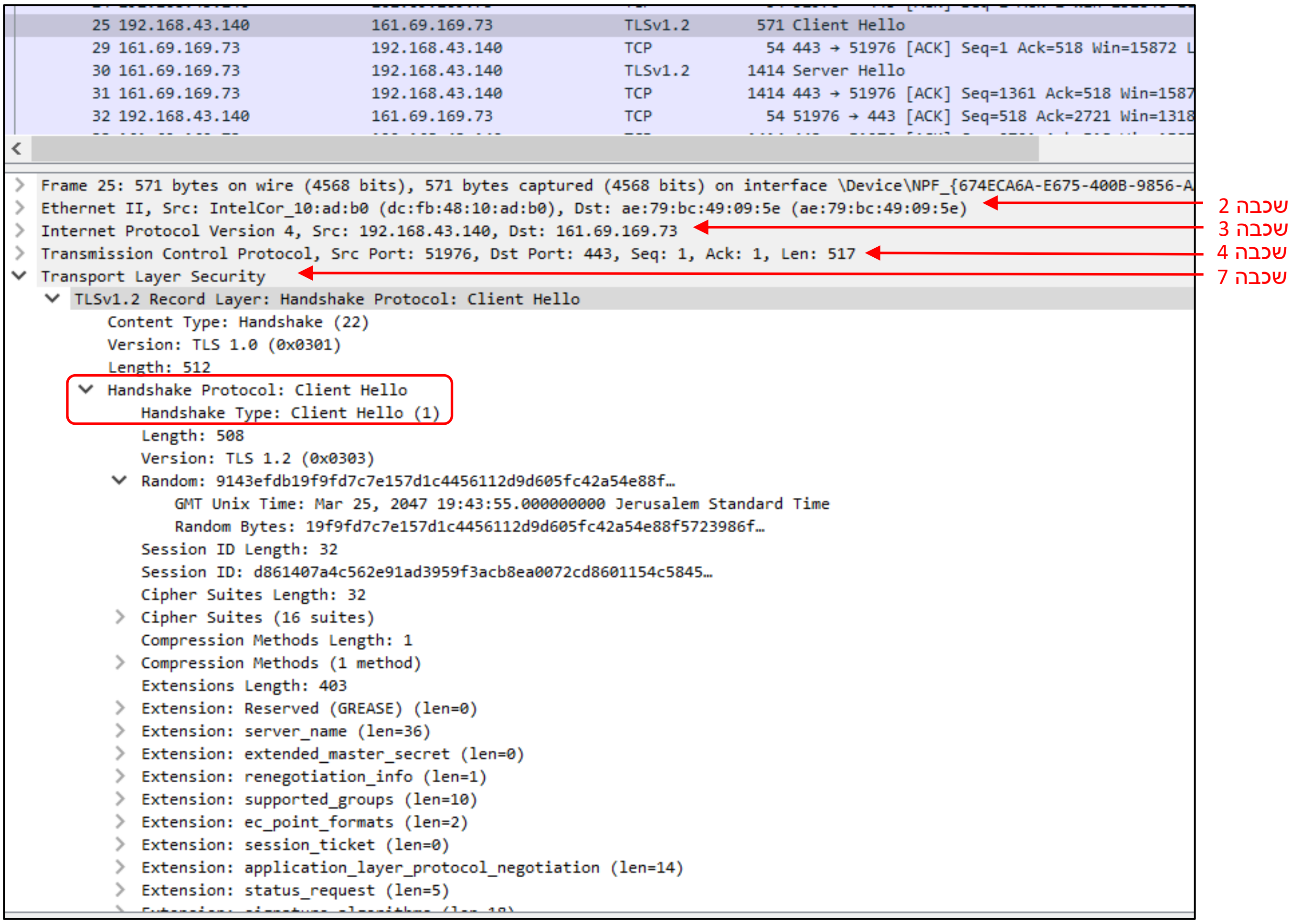


No.	Source	Destination	Protocol	Length	Info
22	192.168.43.140	161.69.169.73	TCP	66	51976 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
23	161.69.169.73	192.168.43.140	TCP	66	443 → 51976 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1360 SACK_PERM=1 WS=512
24	192.168.43.140	161.69.169.73	TCP	54	51976 → 443 [ACK] Seq=1 Ack=1 Win=131840 Len=0
25	192.168.43.140	161.69.169.73	TLSv1.2	571	Client Hello
29	161.69.169.73	192.168.43.140	TCP	54	443 → 51976 [ACK] Seq=1 Ack=518 Win=15872 Len=0
30	161.69.169.73	192.168.43.140	TLSv1.2	1414	Server Hello
31	161.69.169.73	192.168.43.140	TCP	1414	443 → 51976 [ACK] Seq=1361 Ack=518 Win=15872 Len=1360 [TCP segment of a reassembled PDU]
32	192.168.43.140	161.69.169.73	TCP	54	51976 → 443 [ACK] Seq=518 Ack=2721 Win=131840 Len=0
33	161.69.169.73	192.168.43.140	TCP	1414	443 → 51976 [ACK] Seq=2721 Ack=518 Win=15872 Len=1360 [TCP segment of a reassembled PDU]
34	161.69.169.73	192.168.43.140	TLSv1.2	909	Certificate, Server Key Exchange, Server Hello Done
35	192.168.43.140	161.69.169.73	TCP	54	51976 → 443 [ACK] Seq=518 Ack=4936 Win=131840 Len=0
36	192.168.43.140	161.69.169.73	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
37	192.168.43.140	161.69.169.73	TLSv1.2	1076	Application Data
45	161.69.169.73	192.168.43.140	TLSv1.2	328	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
48	161.69.169.73	192.168.43.140	TLSv1.2	440	Application Data
49	192.168.43.140	161.69.169.73	TCP	54	51976 → 443 [ACK] Seq=1666 Ack=5596 Win=131072 Len=0
57	192.168.43.140	161.69.169.73	TLSv1.2	1077	Application Data
67	161.69.169.73	192.168.43.140	TLSv1.2	439	Application Data
78	192.168.43.140	161.69.169.73	TCP	54	51976 → 443 [ACK] Seq=2689 Ack=5981 Win=130816 Len=0
94	192.168.43.140	161.69.169.73	TLSv1.2	1081	Application Data
136	161.69.169.73	192.168.43.140	TLSv1.2	437	Application Data
140	192.168.43.140	161.69.169.73	TCP	54	51976 → 443 [ACK] Seq=3716 Ack=6364 Win=131840 Len=0
2015	192.168.43.140	161.69.169.73	TLSv1.2	1325	Application Data
2027	161.69.169.73	192.168.43.140	TLSv1.2	437	Application Data
2029	192.168.43.140	161.69.169.73	TCP	54	51976 → 443 [ACK] Seq=4987 Ack=6747 Win=131328 Len=0

אפשר לראות בהתחלה את תהליך ה-**TCP handshake** ומיד לאחריו את תחילתו של ה-**TLS handshake**.

הודעה 1 - Client Hello:

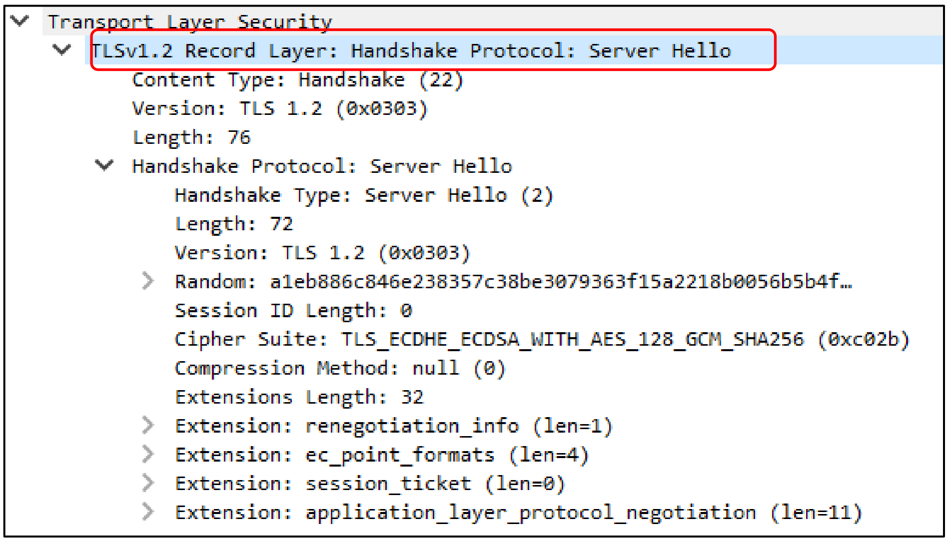
ה-Client מצהיר שהוא מוכן להתחיל בתהליך ה-TLS handshake ושולח כמה פרטים: גרסאות ה-TLS בהן הוא תומך, רשימה של הצפנות (סימטריות) בהן הוא תומך ועוד פרטים טכניים. כל התקשורת בשלב זה נעשית ב-clear text, כלומר ללא הצפנה ולכן נוכל לפתוח ולראות את תוכן הפאקטה של ההודעה (בהמשך לא נוכל). בנוסף, ניתן לראות כי שכבת ה-TLS היא השכבה שעוטפת את כל השכבות מתחתיה.



הודעה 2 - Server Hello + Certificate:

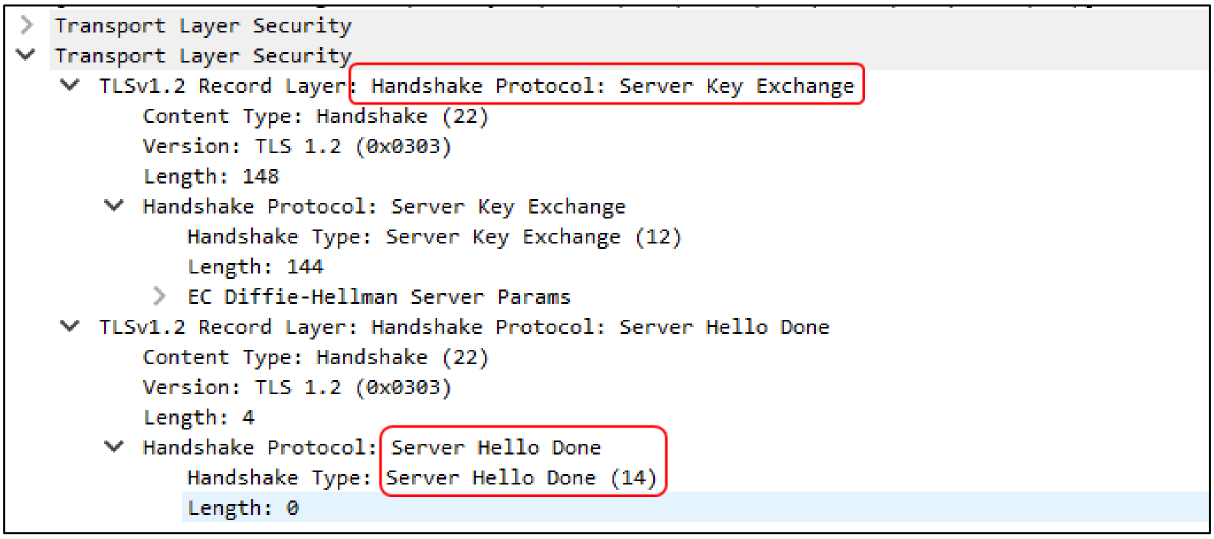
השרת בוחר בגרסת TLS וההצפנה בהן הוא מעוניין לעבוד (למעשה הוא בוחר דיפולטיבית המקסימום שהוא תומך בו) ומצרף את המפתח הציבורי שלו עם Certificate. במקרה שלנו מדובר בפאקטות 30, 31, 33, 34.

<u>פאקטה 30:</u>

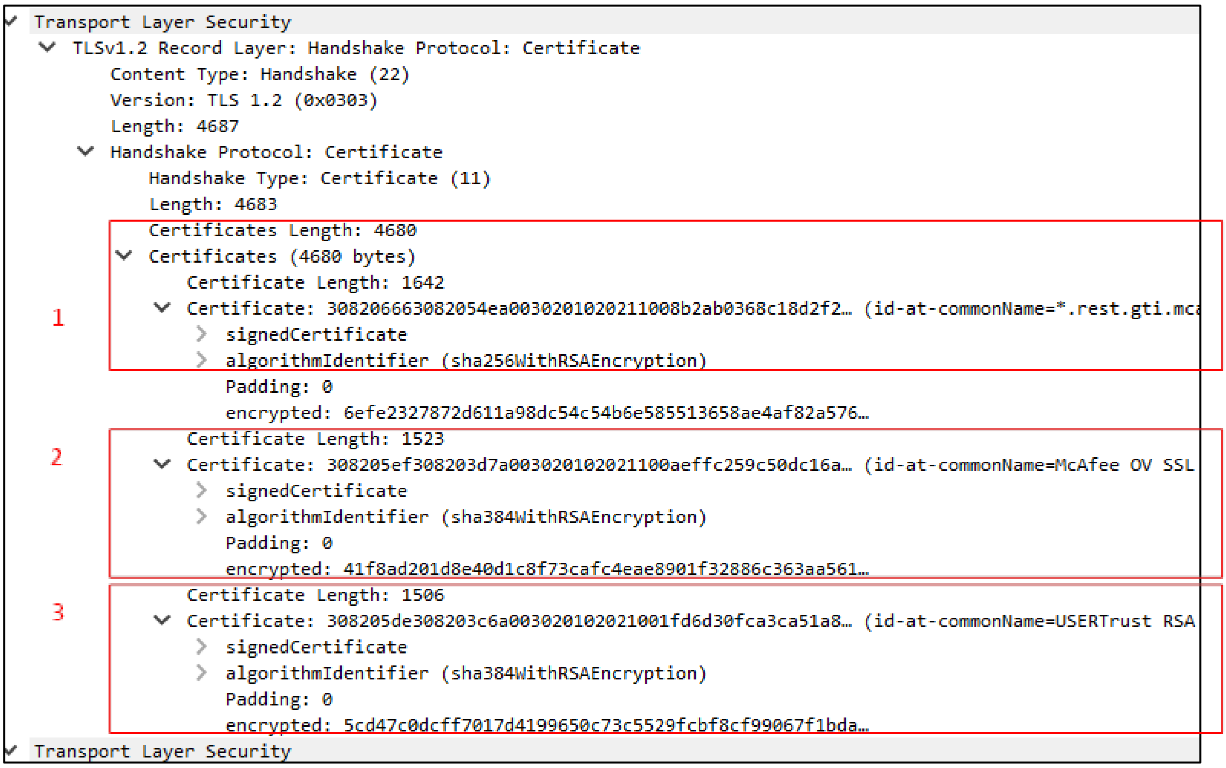


```
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 76
    Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 72
      Version: TLS 1.2 (0x0303)
      Random: a1eb886c846e238357c38be3079363f15a2218b0056b5b4f…
      Session ID Length: 0
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
      Compression Method: null (0)
      Extensions Length: 32
      Extension: renegotiation_info (len=1)
      Extension: ec_point_formats (len=4)
      Extension: session_ticket (len=0)
      Extension: application_layer_protocol_negotiation (len=11)
```

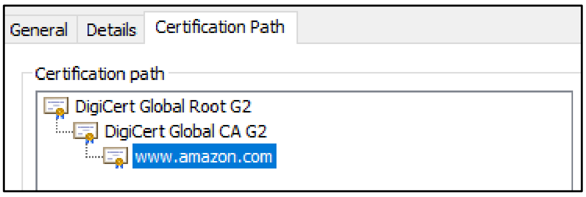
<u>פאקטה 34</u> מכילה את התעודות ואת הודעת Server Done. פה חברינו amazon אומר שהשרת סיים להעביר את המפתחות שלו והוא מוכן לעבודה:



```
Transport Layer Security
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 148
    Handshake Protocol: Server Key Exchange
      Handshake Type: Server Key Exchange (12)
      Length: 144
      EC Diffie-Hellman Server Params
  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
```



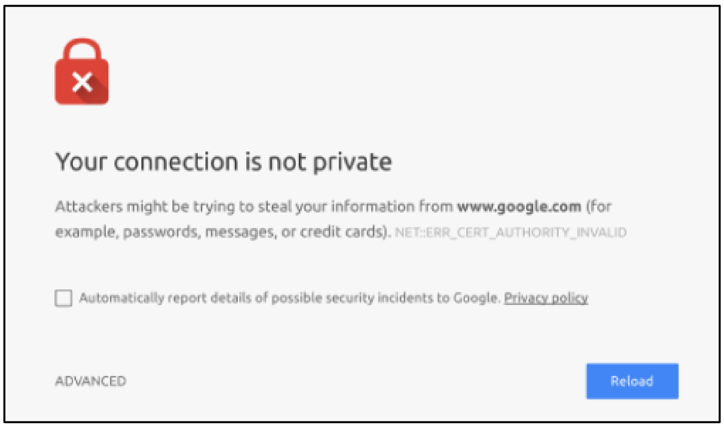
בפאקטה נוכל לראות כי amazon.com מציג לנו 3 תעודות ואת הפרטים לגבי כולן. אם נפתח את התעודה כפי שעשינו בתחילת המסמך נוכל לראות כי אכן מדובר ב-3 תעודות:



כלומר, התעודה של www.amazon.com חתומה על ידי DigiCert Global CA אשר בתורה חתומה על ידי תעודת השורש של DigiCert Global Root - DigiCert. שרשור תעודות קלאסי. אם נלחץ על תוכן התעודות, גם בממשק של חלונות וגם ב-Wireshark נוכל לראות את כל השדות של התעודות עצמן:

בשלב זה למעשה הלוקח (הדפדפן שלנו) מקבל את התעודה מהשרת ובודק מול גורם שלישי שיוכיח שזהו באמת התעודה של amazon.com - האתר שאותו אנחנו רוצים לטעון.

במידה ונרצה לגלוש לאתר שהתעודות שלו לא מהימנות / לא בתוקף, הדפדפן יתריע לנו על כך בצורה הבאה:



יש לציין, שבשלב הזה השרת יכול להחליט שהוא רוצה לאמת בעזרת HTTPS Authentication, את זהות המשתמש (אותנו) ולבקש מה-Client חזרה את ה-Certificate שלו. זה לא נפוץ בכלל כשמדובר על יישומי אינטרנט ובדוגמא לא עלתה שאילתה לכך.

הודעה 3 - העברת מפתח סימטרי: (עכשיו מגיע החלק המעניין שבזכותו עובד HTTPS)

לאחר שהדפדפן שלנו אימת את התעודה של השרת של amazon.com הוא מייצר ג'יבריש **אקראי** (מאוחר יותר זה יהיה המפתח הסימטרי לתקשורת) לצורך ה- session הנוכחי עם השרת. את אותו ג'יבריש הדפדפן מצפין **בעזרת המפתח הציבורי של השרת עצמו** ושולח לשרת. השרת מנגד, שמחזיק במפתח הפרטי, מפענח את ההצפנה של המפתח הציבורי שלו וכעת גם לו וגם לדפדפן יש ג'יבריש aka סוד משותף - מפתח סימטרי זהה.

```
Transport Layer Security
  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: 040c9a2ce5e4d7e98d600fce94a140c647e924189eb56387…
  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
  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
```

מכאן ואילך, הדפדפן והשרת יכולים לשלוח מה שבא להם ולהשתמש באותו ג'יבריש בשביל להצפין את התעבורה. אבל רגע, איך הם מסנכרנים על זה ששניהם יודעים את הסוד?

הודעה 4 - שימוש במפתח הסימטרי לצורך אימות וסיום תהליך ה-TLS handshake:

בשלב זה השרת משתמש במפתח הפרטי שלו כדי לפענח את ההודעה הקודמת, לחלץ את המפתח הסימטרי - ולעבור להשתמש בו. ההודעה הראשונה שנשלחת ל-Client היא הודעת Finish שמוצפנת במפתח הסימטרי שה- Client עצמו סיפק. הודעה שעוזרת לוודא שתהליך העברת המפתח הסימטרי והשימוש בו ע"פ ההצפנה הנכונה - הצליח. כהוכחה לכך אפשר לראות ש- wireshark עבר לסווג את התקשורת בינינו כ- "Application Data", שבשפה עממית אומר - "וואלה אין לי מושג מה יש שם אחי, זה מוצפן".

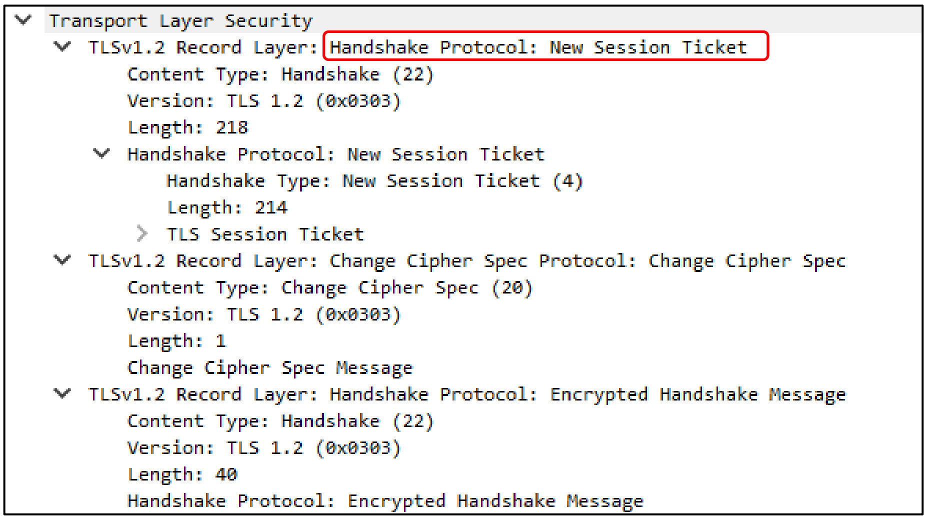
```
Transport Layer Security
  TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
    Content Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 1017
    Encrypted Application Data: 0000000000000001565d6a8775f572e4e99813d34d71b8b9…
```

<u>אנקדוטה מעניינת:</u>

הדפדפן פותח מספר connections במקביל מול כל שרת בכדי להאיץ את ההורדה. על מנת לא להגיע למצב בו משלמים את התקורה של TLS handshake מספר רב של פעמים (יכול להגיע עד שנייה כשמדובר על חיבור למשל בין ישראל וארה"ב), נוספה יכולת בשם **Session Ticket** (תקן [RFC 5077](RFC 5077)) המאפשרת ל-connection שני, שלישי וכך הלאה, מול אותו השרת ולעשות שימוש חוזר במפתח הסימטרי, אימות, הסכמה על הצפנות וכו'.

עם הודעת ה-Finish השרת שולח Session Ticket (מוצפן) לClient. כל connection חדש יכול לצרף את ה-ticket הזה בכדי "להצטרף" ל-session HTTPS שמחזיק השרת - ולחסוך לעצמו תהליך handshake נוסף.

בניגוד ל-HTTP שבברירת המחדל הוא stateless (מתנתק אחרי כל response) פרוטוקול HTTPS הוא תמידstateful (במסגרת ה-session). החיבור במקרה של הדוגמא שלנו:

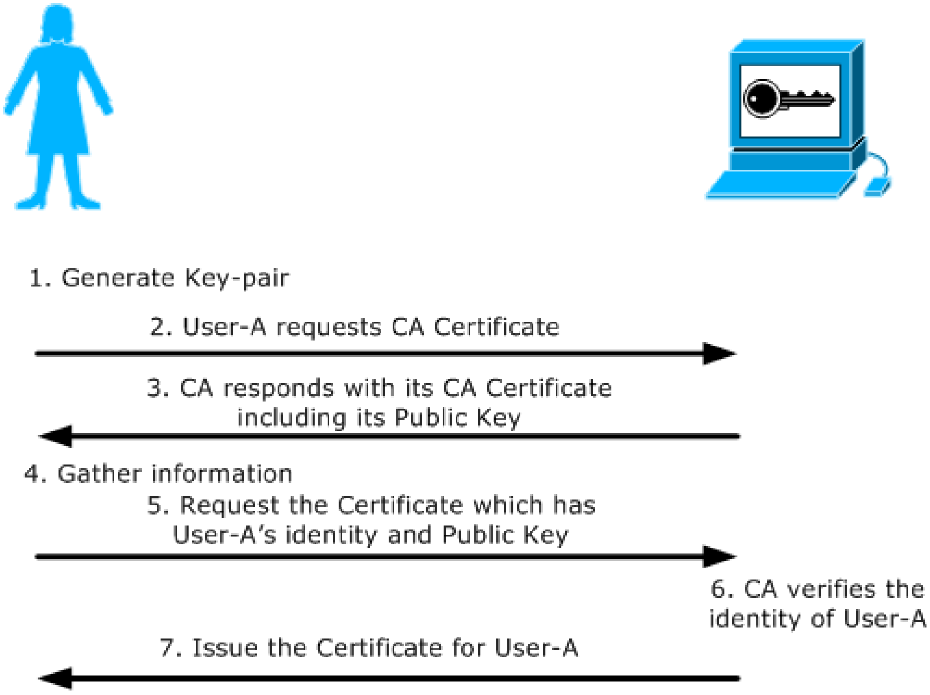


אז איך חותמים תעודת X.509?

תהליך לקבלת תעודה דיגיטלית

אבל רגעעע, אני רוצה להקים אתר ושהתעודות שלו יהיו טובות. איך אני עושה את זה? אז ראשית, ה-vendor (GoDaddy ושות') שמארח את האתר שלכם בדר"כ גם יגבה מחיר על השימוש ב-ssl, אז אין צורך לדאוג מזה. אבל אם בכל זאת מעניין אתכם כיצד מתרחש התהליך לרכישת תעודה, הרי הוא לפניכם.

האיור הבא מציג את הדרך לקבלת תעודה בצורה מופשטת:



אופן התהליך:

1. **השגת צמד מפתחות:** משתמש קצה מקבל\ מייצר צמד מפתחות (פרטי וציבורי) באופן בטוח
2. **בקשת הכרה מ-CA:** המשתמש מבקש תעודה משרת CA.
3. **תשובת הכרה להזדהות מ-CA:** ה-CA מציג שהוא אכן מי שהוא טוען שהוא באמצעות שליחת תעודה עם המפתח הציבורי והחתימה הדיגיטלית שלו (שנחתמה באמצעות המפתח הפרטי שלו).
4. **איסוף מידע:** המשתמש אוסף את כל האינפורמציה ששרת ה-CA דורש ממנו על מנת להוכיח את זהותו. מידע זה יכול לכלול את האימייל, מידע משפטי ופרטים אישיים נוספים של המשתמש.
5. **שליחת בקשת רישום תעודה:** המשתמש שולח בקשה לרישום תעודה על שמו. הבקשה מכילה את המפתח הציבורי של המשתמש והיא נחתמת עם המפתח הציבורי של ה-CA.
6. **אימות המשתמש על ידי ה-CA:** ה-CA בוחן את ההודעה לבקשת תעודה של המשתמש ומאמת את זהותו. לאחר מכן הוא יוצר תעודה וקושר את זהותו של המשתמש לתעודה (אצלו במערך הנתונים).
7. **שליחת התעודה:** ה-CA שולח למשתמש את התעודה חתומה.

אז איך כל זה נראה בפרקטיקה?

דיברנו המון על איפה תעודות נמצאות במחשב, מהם השדות שמרכיבים אותן, כיצד מעבירים אותן ובכללי איך נראה כל התהליך של חתימה עליהן. אבל זה תיאורטי בלבד. בתת-פרק הקרוב נראה כיצד ניתן לחתום בעצמנו על תעודה (מימוש self-signed certificate), כיצד אנחנו יכולים להפוך למיני CA ולחתום על תעודות שאנחנו מפיקים ולשנע אותן למי שחפץ.

ספריה \ כלי שנעשה בו שימוש בשביל לממש את זה בעצמנו הוא **OpenSSL**. התוכנה היא ערכת כלים חזקה, מסחרית עם המון פיצ'רים עבור פרוטוקול SSL\TLS. openssl עוסק הרבה בסקירה וביצירה של תעודות ומפתחות.

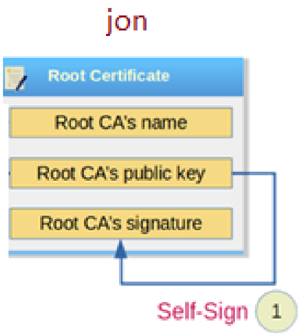
הערה: מטעמי נוחות בלבד, בדוגמאות הקרובות השתמשתי לא מעט ב-openssl בקלי לינוקס, התוכנה המקבילה אליה ב-windows היא certutil ואפשר להוריד ולהשתמש ב-openssl גם ב-windows.

self-signed certificate

בסעיף הקרוב נייצר 3 קבצים (תעודות ליתר דיוק):

- **server.key** ⇒ Private Key
- **server.csr** ⇒ Certificate Signing Request
- **server.crt** ⇒ Self-signed certificate

כאשר באמצעותם נסיים את הסעיף כשברשותנו המבנה הבא:



ראשית נייצר את המפתח הפרטי של מחזיק התעודה:

```
jon@kali:~/Documents/certstuff$ openssl genrsa -out server.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
..........................................................................................+
+++
..........................................................................................
..........................................................................................
..++++
e is 65537 (0×010001)
```

openssl genrsa היא מטודה ליצירת מפתח RSA. ‏4096 הוא גודל המפתח. אם נרצה לבחון את המפתח שיצרנו, נריץ:

```
jon@kali:~/Documents/certstuff$ openssl rsa -in server.key -noout -text
```

ונקבל:

```
RSA Private-Key: (4096 bit, 2 primes)
modulus:
    00:e9:78:14:be:6a:50:93:4b:66:20:38:10:17:e3:
    30:9f:53:e0:b2:a5:98:8b:12:4c:87:25:c1:ec:2d:

publicExponent: 65537 (0x10001)
privateExponent:
    00:e2:d4:29:0b:4b:8d:97:5d:e5:0e:1e:89:fc:3b:
    5e:0d:1e:58:2c:19:59:58:9c:e1:bf:58:7e:cd:70:

prime1:
    00:ff:df:3a:70:52:5c:2d:a4:40:ec:b7:ae:8d:9d:
    2a:01:b5:17:f0:3f:ad:56:10:ae:4d:69:62:e2:10:

prime2:
    00:e9:95:fb:c1:66:70:ab:cf:8a:c4:3f:e3:94:73:
    8d:80:a9:fb:a6:6a:27:a2:93:fe:68:86:1c:91:ff:

exponent1:
    2d:fe:ed:dd:65:20:c6:df:a1:00:2c:a0:c7:3a:3a:
    63:e6:2e:f0:74:31:ef:a7:bd:fd:92:75:41:18:36:

exponent2:
    00:a4:8c:32:70:58:50:b0:c7:a0:52:85:4b:35:6d:
    7d:56:10:e8:a4:9f:8d:52:3a:fc:f1:f6:df:b3:7d:

coefficient:
    00:c6:c9:44:be:9b:c4:9c:d0:8e:e6:8b:8d:e2:fe:
    6b:77:98:e6:c0:2e:c4:cb:a6:7a:6e:c4:d5:1d:fe:
```

בפלט למעלה אפשר לראות את המספרים הראשוניים שמרכיבים את המפתח RSA הפרטי שיצרנו. המפתח הפרטי חייב להישמר בסוד ולהיות זמין רק לישות שמאשרת. כלומר, אין לשלוח אותו לאף אחד אחד, **כולל לא ל-certificate issuer.**

נגיש בקשה לחתימה על התעודה - (**CSR**) *Certificate Signing Request* בלעז. יהיה עלינו להזין מספר שדות בנוגע לזהותנו, רובם לא רלוונטיים והתוכן שנזין בהם לא יהיה חשוב. השדה היחיד שיש לשים לב אליו הוא ה-Common Name וזה מכיוון שבאמצעותו מתבצע הזיהוי.

```
jon@kali:~/Documents/certstuff$ openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IL
State or Province Name (full name) [Some-State]:Israel
Locality Name (eg, city) []:Tel Aviv
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IDF
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:jon
Email Address []:

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

נבחן את תוכן ה-CSR עם openssl:

```
jon@kali:~/Documents/certstuff$ openssl req -in server.csr -noout -text
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IL, ST = Israel, L = Tel Aviv, O = IDF, CN = jon
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                    00:e9:78:14:be:6a:50:93:4b:66:20:38:10:17:e3:
                    30:9f:53:e0:b2:a5:98:8b:12:4c:87:25:c1:ec:2d:

                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         9e:18:40:b2:cb:c2:35:88:86:12:9b:d8:3b:70:f3:55:64:4c:
         ce:51:71:60:30:cb:73:9e:f1:e8:99:50:e5:ee:8c:11:d2:45:
```

לאחר מכן, ניקח את הבקשה שיצרנו לחתימה ונחתום עליה (עם אותו מפתח שלנו):

```
jon@kali:~/Documents/certstuff$ openssl x509 -req -days 365 -in server.csr -signkey server.key
-out server.crt
Signature ok
subject=C = IL, ST = Israel, L = Tel Aviv, O = IDF, CN = jon
Getting Private key
```

נקבל את הקובץ server.crt - זוהי התעודה שיצרנו וחתמנו עליה.

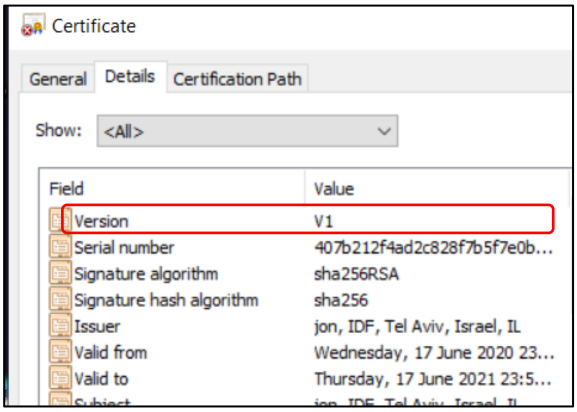
אם נבחר להביט גם על התוכן של הקובץ שלה נוכל לראות שנוסף תאריך פג תוקף לתעודה, המנפיק שלה (אנחנו) והנמען שעברו הנפיקו את התעודה (גם אנחנו - מכאן השם self-signed certificate):

```
jon@kali:~/Documents/certstuff$ openssl x509 -noout -text -in server.crt
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            40:7b:21:2f:4a:d2:c8:28:f7:b5:f7:e0:b9:e9:4e:5a:1e:3f:09:56
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IL, ST = Israel, L = Tel Aviv, O = IDF, CN = jon
        Validity
            Not Before: Jun 17 20:55:47 2020 GMT
            Not After : Jun 17 20:55:47 2021 GMT
        Subject: C = IL, ST = Israel, L = Tel Aviv, O = IDF, CN = jon
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                    00:e9:78:14:be:6a:50:93:4b:66:20:38:10:17:e3:
                    30:9f:53:e0:b2:a5:98:8b:12:4c:87:25:c1:ec:2d:

                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         71:c3:04:61:07:9d:4c:87:ef:5d:f7:fb:5e:03:b1:cb:9b:68:
         ea:fb:79:d9:da:6a:e0:f2:34:a3:86:13:f6:86:f6:f9:df:9c:
```

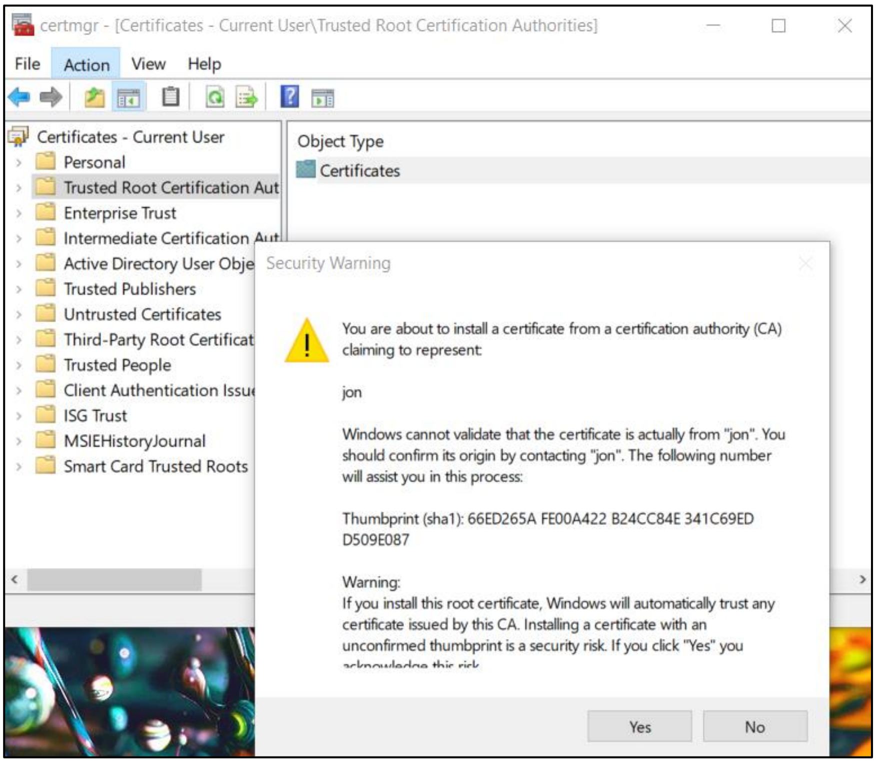
ו.. מזל טוב, אנחנו עכשיו CA. בערך ☺

על מנת להיות "CA אמיתי" אנחנו צריכים ללכת ולצרוב את תעודת השורש שיצרנו על פני כל המחשבים בעולם. זה טיפה שאפתני. לצרכי ארגון שבסך הכל רוצה להנגיש אתרי HTTPS למכשירים שברשותו כל שצריך לעשות זה להטמיע פעם אחת את התעודה בכל המכשירים שיש לארגון (עד שפג התוקף של התעודה כמובן).

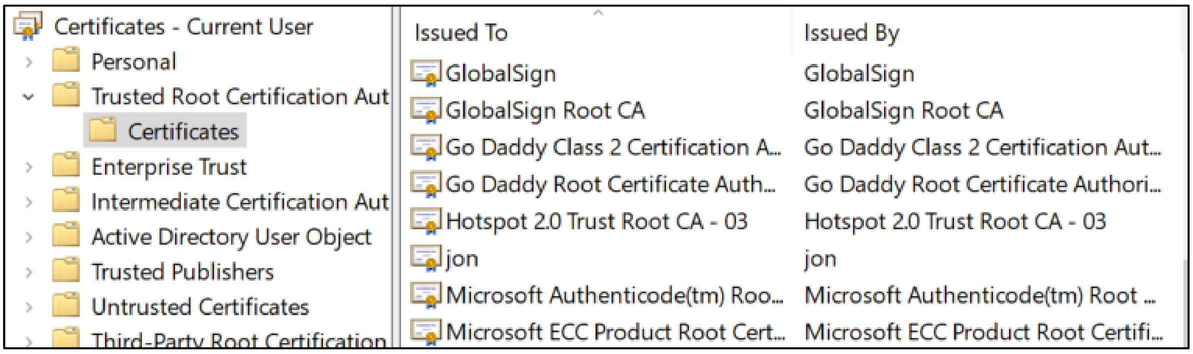
יש לציין כי במקרה שמוצג בסעיף הנ"ל לא נעשה שימוש ב-extensions ביצירת התעודה. על מנת להוסיף הרחבות לתעודה יש צורך להשתמש בסכמות של **openssl.config**. זה לא מורכב לביצוע אך זה פחות רלוונטי מכיוון שעיקר מטרת הסעיף היא להראות שימוש\ ניהול של בתעודות (בסעיף הבא נדגים שימוש בסכמה ליצירת תעודה v3). לכן, התעודה שיצרנו כעת היא למעשה X.509v1:



במידה ונרצה להפיץ את התעודה שלנו לכל המכשירים שברשותנו (לדוגמא - מכונת Windows) נצטרך לעשות import לתעודה לתוך ה-Certificate Store של המכשיר:



מערכת ההפעלה כמובן ישר תצעק עלינו מכיוון שאנחנו הולכים להגדיר במחשב שלנו תעודה (מאוד לא בטוחה שהמערכת לא יכולה לוודא את זהות ה-CA) כ-trusted **root** certification authenticators - מקום שרק מספר מצומצם מאוד של תעודות (מאומתות ובטוחות מאוד) מגיעות אליו. Welp, זה לצורכי מחקר.



עכשיו אנחנו באמת root CA על המכשיר שלנו.

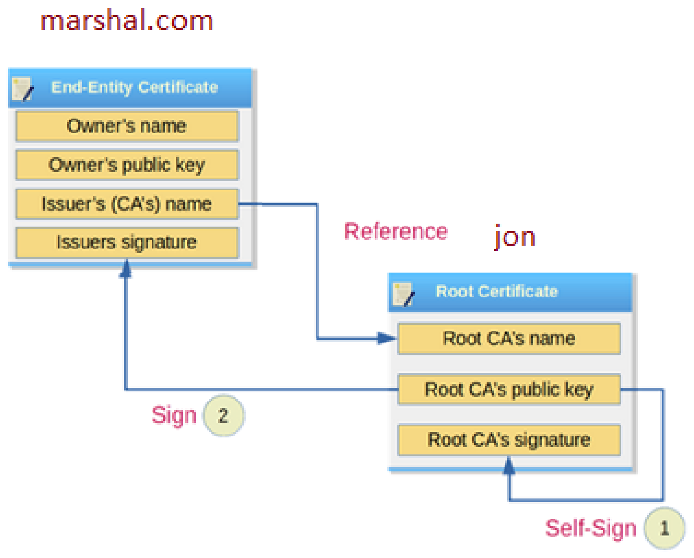
כך למעשה אנחנו יכולים לעבור על פני כל המכשירים שיש לנו בארגון ולהטמיע את התעודה שלנו.

השלב הבא הוא להשתמש בכוח שלנו כ-CA על פני כל המכשירים שברשותנו ולחתום על תעודות נוספות על מנת לאפשר למכשירים לעשות שימוש ב-HTTPS **שאנחנו מאשרים**.

Creating CA-Signed Certificates

נגיד אנחנו מנהלים אתר גדול לרישום לקורסים בארגון שלנו. שם האתר - marshal.com. במידה ונרצה שהאתר יתמוך ב-HTTPS נצטרך להנפיק עבורו תעודה (באמצעות root CA שיצרנו בסעיף הקודם) על מנת שיוכל להציג אותה בעת חיבור ה-TLS בפונקציית server hello() ללקוח שרוצה להתחבר.

יאללה לעבודה. המבנה שניצור בסעיף הקרוב:



ראשית, בדומה ליצירת תעודת ה-CA, ניצור מפתח פרטי ל-marshal.com:

```
jon@kali:~/Documents/certstuff$ openssl genrsa -out marshal.com.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
..................................................+++++
.................................................................................+++++
e is 65537 (0x010001)
```

שנית, ניצור את ה-CSR (כאמור גם פה יהיה עלינו להכניס מידע מזהה שיקשור את זהות האתר למפתח הפרטי שלו):

```
jon@kali:~/Documents/certstuff$ openssl req -new -key marshal.com.key -out marshal.com.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IL
State or Province Name (full name) [Some-State]:Israel
Locality Name (eg, city) []:Tzrifin
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:marshal
Email Address []:

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

בשלב זה, במידה והיה מדובר ב-CSR (בקשה לתעודה) סטנדרטית, היינו שולחים רק את הבקשה (קובץ marshal.com.csr) ל-CA גדול כגון Verisign, משלמים לו כמה דולרים, השרת שלו היה עובר על ה-CSR ששלחנו ומאשר / לא מאשר את התעודה שלנו באמצעות חתימה עם תעודת השורש שלו על התעודה שלנו. (בפישוט יתר).

אבל - **אנחנו מחזיקים ב-trusted root certification על כל המכשירים שלנו!** במידה ואנחנו רוצים לעבוד **מפנים** הארגון הפרטי שלנו עם HTTPS, אנו כבר לא חייבים לשלם כסף בשביל שמישהו אחר יחתום לנו על האישור - אנחנו יכולים לחתום בעצמנו!

איך עושים את זה:

מגרסה 58 של Google Chrome שונה ה-setup של הדפדפן והוא אינו מאפשר יותר לעשות שימוש ב"Add Exception" והוא לא מאפשר לעשות שימוש יותר בשדה CommonName בתעודה ובמקום זה הוא בודק מספר שדות בהרחבה. כלומר, תעודה "פשוטה" (v1) לא כשרה יותר והיא לא תוכל לספק לנו גלישת HTTPS.

על מנת להתגבר על כך, נכתוב את הסכמה CA.ext וניצור תעודות עם שדות של extensions:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = dev.mergebot.com
DNS.2 = dev.mergebot.com.192.168.1.19.xip.io
```

[זה פחות רלוונטי להצגה עצמה כיצד הנדסנו את השדות עצמם, אבל זה בהחלט לא מסובך ואפשר ללמוד על זה בכמה דק' באינטרנט]

נשתמש במפתח הפרטי של ה-CA, בתעודת ה-CA ובקובץ הקונפיגורציה על מנת להנפיק תעודה ל-CSR שאתר marshal.com ביקש:

```
jon@kali:~/Documents/certstuff$ openssl x509 -req -in marshal.com.csr -CA server.crt -CAkey server.key -CAcreateserial -out marshal.com.crt -days 180 -extfile CA.ext
Signature ok
subject=C = IL, ST = Israel, L = Tzrifin, O = Internet Widgits Pty Ltd, CN = marshal
Getting CA Private Key
```

נבדוק שאכן קיבלנו תעודה כשרה למהדרין:

```
jon@kali:~/Documents/certstuff$ openssl x509 -noout -text -in marshal.com.crt
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number:
         1c:ea:b4:37:8c:45:fe:ee:53:f7:c5:03:74:16:fe:b8:df:50:be:64
      Signature Algorithm: sha256WithRSAEncryption
      Issuer: C = IL, ST = Israel, L = Tel Aviv, O = IDF, CN = jon
```

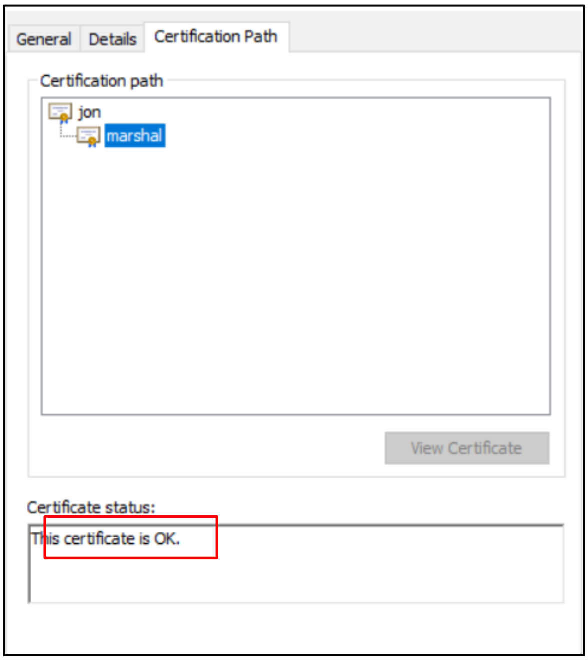
```
      Validity
         Not Before: Jun 17 22:55:15 2020 GMT
         Not After : Dec 14 22:55:15 2020 GMT
      Subject: C = IL, ST = Israel, L = Tzrifin, O = Internet Widgits Pty Ltd, CN =
marshal
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
            RSA Public-Key: (2048 bit)
            Modulus:
               00:f3:50:08:f4:5e:d7:94:aa:57:f4:1b:6e:1d:b1:
               2a:f5:80:b4:ab:77:86:03:b3:1e:c9:1c:9d:5e:a4:

            Exponent: 65537 (0x10001)
      X509v3 extensions:
         X509v3 Authority Key Identifier:
            DirName:/C=IL/ST=Israel/L=Tel Aviv/O=IDF/CN=jon
            serial:40:7B:21:2F:4A:D2:C8:28:F7:B5:F7:E0:B9:E9:4E:5A:1E:3F:09:56

         X509v3 Basic Constraints:
            CA:FALSE
         X509v3 Key Usage:
            Digital Signature, Non Repudiation, Key Encipherment, Data Encipherment
         X509v3 Subject Alternative Name:
            DNS:marshal.com, DNS:dmarshal.com.192.168.1.19.xip.io
   Signature Algorithm: sha256WithRSAEncryption
       17:c3:ce:61:79:a1:82:3a:5e:d1:f9:44:50:ff:8e:80:bf:ff:
       72:9b:4a:9d:a4:03:fb:bb:78:cf:aa:96:b3:a6:38:7f:44:b9:
```

מדהים, תעודה מן המניין.

שימו לב שכעת מדובר בתעודת X.509v3. כמו כן, ניתן לראות כי הגדרנו שדות DNS שיכילו את התעודה שלנו גם כן. החלק החשוב שיש לשים אליו לב הוא שדה ה-issuer:

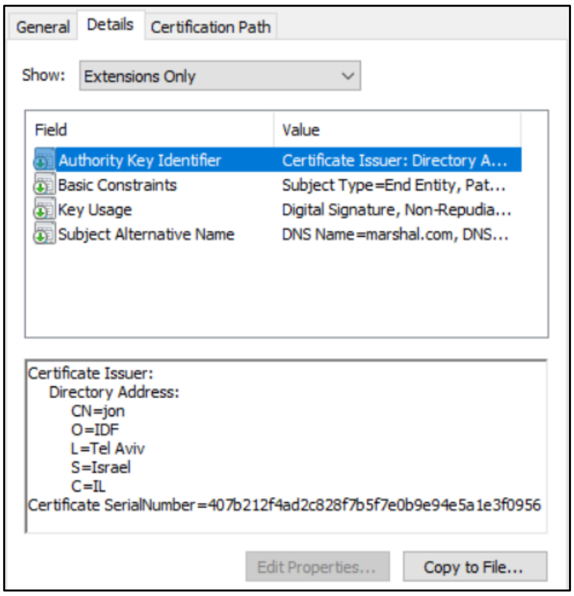
Issuer: C = IL, ST = Israel, L = Tel Aviv, O = IDF, CN = jon

כאשר אתר marshal.com יציג למשתמש הקצה את התעודה, הדפדפן של המשתמש יבדוק האם הוא סומך על תעודות ש-jon ערב להן ויאשר את כניסתו של המשתמש לאתר:



אפשר לראות שהתעודה של marshal חתומה על ידי jon ומערכת ההפעלה מזהה כי מדובר בתעודה ולידית.

נסתכל על ההרחבות של התעודה:



כעת marshal.com יוכל להטמיע את התעודה בתור תעודת צד שרת ולהציג אותה לכל המכשירים של הארגון שלנו כשהם יגלשו אליו ויפתחו בתהליך TLS 4 way handshake.

אבטחת תעודות X.509

נפתח עם הפיל שבחדר - קיימות חולשות בתקן הנובעות מאבני הבניין שמרכיבות אותו:

חולשות ארכיטקטורות:

- **שימוש ב-blacklist לתעודות לא מורשות** (CRL, OCSP). פיתוח מאובטח תמיד ישים דגש על שימוש נכון ב-whitelist עקב הנחת הבסיס שתמיד יהיו פרצות \ מקרי קצה שהמפתח לא חזה ולכן תמיד תהיה עדיפות להימנע משימוש ב-blacklist כאשר מדובר בהחרגות.

  במקרה של התעודות, אם לקוח מסתמך על תעודות **רק** כאשר הCRLים זמינים עבורו לבדיקת התעודות, הוא מאבד את יכולת התפעול בתצורת offline. אי לכך, רוב ה-clients מתוכנתים לסמוך ולעבד תעודות גם כאשר אין להם גישה ל-CRL. אין צורך להעמיק במחשבה יתרה בשביל לדמיין סיטואציה בה תוקף פוטנציאלי ששולט על ערוץ התקשורת, מחליט לבטל את ה-CRLים. אנאלוגיה שמתאימה לסיטואציה הזו היא חגורת בטיחות שפועלת תמיד כשורה אלא כשמתרחשת תאונה.
- **אחסון** - CRL-ים נוטים להיות גדולים מאוד וחסרי סדר סטנדרטי.
- אין מענה בתקן למצב בו נעשה ביטול ל-root certificates.

- **שילוב מידע** - מידע על זהות, תכונות ומדיניות מקובצים בתוך container אחד ויחיד. מבנה זה מעלה בעיות פרטיות, מיפוי ומדיניות תחזוקה.
- **בעיות סמכותיות** - CAs לא יכולים להגביל intermediate CAs מלהנפיק תעודות מחוץ ל-namespace מסוים וסופי. לפיכך, קיים מספר גדול CAs באינטרנט ולסווג אותן ואת המדיניות שלהן היא משימה לא ריאלית.
- **זמן חישוב** - כתוצאה משרשור תעודות, הצלבת חתימות (cross-signing) ומספר CAs גדול, תהליך ווידוא של תעודה יכול להסתבך ולהיות יקר מבחינת זמן חישוב.

בעיות עם CAs:

- הישות שמחזיקה במפתח הפרטי (ה-subject), ולא הצד הנשען שעושה שימוש בתעודות, רוכשת את התעודות עצמן. ה-subject לרוב יבחר במנפיק הזול ביותר ושירותי האבטחה יהיו הראשונים להיפגע מכך.
- כמו כל העסקים, חברות CA כפופות לתחומי השיפוט החוקיים שבהם הן פועלות, ועשויות להיות כפופות מבחינה משפטית לפגוע באינטרסים של הלקוחות והמשתמשים בהן .סוכנויות הביון עשו ועושות שימוש בתעודות כוזבות שהונפקו באמצעות התחייבות חוץ-חוקית של CAs.
- CAs רבים לא מספקים כל סוג של אחריות למשתמשים בנוגע לתעודות שהם עצמם הנפיקו.

בעיות במימוש:

היישום בפועל של תעודות X.509 סובל מפגמים בעיצוב, באגים, פרשנויות שונות לתקנים, קושי בהטמעה וחוסר יכולת פעולה הדדית של סטנדרטים שונים. כמה בעיות כוללות:

- מימושים רבים מבטלים את הבדיקה של גניזת (revocation) התעודות מכיוון שהיא מקשה על תפעול המערכת / הרשת ומעמיסה עלייה רבות.
- DNs מהווים חלק אינטגרלי מהתקן והם קשים להבנה ומימוש נכון (עקב חוסר גלובליזציה בנושא ואי התאמה במינוח).
- למרות חשיבותם, Policy Constraints ו-Name Constraints לרוב לא נתמכים.
- אכיפה של OIDs שנוצרו בהתאמה אישית קשה לביצוע.
- אורך לא מוצהר של תכונה (שדה) עשוי להגביל מוצר ספציפי.

חולשות קריפטוגרפיות:

חתימות דיגיטליות מסתמכות על פונקציות hash (כמו MD2) קריפטוגרפיות מאובטחות. כאשר PKI מאפשרת להשתמש בפונקציית hash שאינה מאובטחת, תוקף יכול לנצל חולשות בפונקציית ה-hash כדי לזייף תעודות. באופן ספציפי, אם תוקף מסוגל לייצר hash collision הוא יכול לשכנע CA לחתום על תעודה עם שדות רגילים, כאשר ה-hash של אותם תכנים זהה ל-hash של תעודה זדונית, שנוצרה על ידי התוקף עם ערכים שהוא בחר בקפידה ובכך ליצור תעודה חתומה על ידי CA שמכילה שדות פגיעים כמו תוקף זמן שונה, hostname שונה וכו'.

הערה: תקופה שכנראה חלקינו זוכרים היא התקופה בה דפדפנים כגון Chrome, Firefox, Edge, Safari החלו לדחות תעודות שעשו שימוש ב-SHA-1 (מכיוון שהוכח POC שמאפשר לרוורס את ה-hash) ולכן הדפדפן היה מקפיץ התראות על שימוש לא בטוח גם ב-HTTPS. סוגיה שלא נפתרה עד היום היא שהמון מערכות אזרחיות\לא אזרחיות עדיין מאפשרות תמיכה מלאה בSHA-1. תמיכה בשיטות גיבוב חלשות מתאפשרות מכיוון שלא קיימת דרך לאכוף ברמת התקן את השימוש בשיטות hash והאכיפה נשארת למערכות עצמן בלבד.

המלצות לאבטחת TLS בצורה נכונה

התת פרק הקרוב מכוון להעניק הדרכה להטמעת TLS באופן נכון באפליקציה / מערכת העושה בו שימוש. לפני שאתם ממהרים לקרוא, אני רק רוצה לכסת"ח בקטנה את עצמי ואת כל העולם ולהצביע שוב לכותרת של

הפרק - "המלצות". אני ו/או מערכת המגזין לא אחראיים בשום צורה לבעיות שיהיו במידה ותבחרו להטמיע את ההמלצות שבפרק הנ"ל (למרות שוואלה הן אחלה סטנדרט מינימאלי).

מכל מקרה, מומלץ גם למי שלא הולך לממש TLS אצלו בבית לקרוא את הפרק הקרוב מכיוון שהוא מלמד ומעניין.

פרוטוקול Challenge-Response:

פרוטוקול Challenge-Response הוא פרוטוקול אימות זהויות, שבו ישות אחת מוכיחה את זהותה לישות אחרת בהפגנת ידיעת סוד המשויך לאותה ישות מבלי לחשוף את הסוד עצמו במהלך הפרוטוקול.

על מנת לזהות שאנו מתקשרים עם הישות הנכונה בצד השני לא מספיק לעבור על תעודה שהיא שולחת בלבד. חובה לבצע שימוש באתגר-מענה גם כן.

הרעיון הבסיסי של פרוטוקול Challenge-Response לאימות זהויות מאוד פשוט, במקום לשלוח את הסיסמה עצמה באופן גלוי, הסיסמה **משמשת כמפתח הצפנה** לאחר המרה למפתח הצפנה באמצעות

פונקציה מוסכמת (בדרך כלל פונקציית גיבוב קריפטוגרפית), כך שהנתונים שעוברים בערוץ התקשורת אינם במצב קריא.

השימוש במפתחות הצפנה שנגזרים מסיסמת הלקוח כדי לאמת את זהותו נקרא **אימות חזק**. בפשטות - הלקוח שולח אתגר לשרת בו הוא מבקש שיצפין לו איזה string רנדומאלי עם המפתח הפרטי של השרת ואז הלקוח (באמצעות המפתח הציבורי שנמצא בתעודה) מפרק את הטקסט המוצפן ומשווה אותו לאותו string. מה שמוכיח באופן חד משמעי שעל מנת להצפין את אותו string השרת חייב להחזיק במפתח הפרטי - מה שמאשר את זהותו.

ניתן ליישם פרוטוקול Challenge-Response במספר דרכים: באמצעות טכניקות סימטריות בלבד, בשילוב הצפנה א-סימטרית או בשיטת הוכחה באפס ידע.

חלה חובה לממש לפחות פרוטוקול אתגר-מענה אחד במסגרת מימוש TLS.

יישום פרוטוקול אתגר-מענה המבוסס על חתימה דיגיטלית

מנגנון X.509 מיישם פרוטוקול אתגר-מענה המבוסס על חתימה דיגיטלית. פרוטוקולים אלו מסתמכים על ההנחה כי המאמת מסוגל להשיג את מפתח האימות של החתימה הדיגיטלית של המוכיח באמצעים מקובלים (המבטיחים את שייכותם למשתמש) כמו באמצעות סרטיפיקט מפתח פומבי. להלן 2 דוגמאות לפרוטוקולים מסוג זה עבור מנגנוני האימות הסימטריים.

אימות חד-צדדי מבוסס חתימה דיגיטלית עם חותם זמן:

המוכיח שולח חותם זמן החתום בחתימה הדיגיטלית שלו. בקבלת המסר החתום, המאמת B יכול לוודא באמצעות המפתח הפומבי של A כי החתימה אכן נעשתה על ידו ובזה הוא מקבל את זהותו. על המאמת כמובן להשיג מראש את מפתח האימות הפומבי של החותם כאמור, באמצעים שיבטיחו עבורו את שלמותו ושייכותו לחותם.

פרוטוקול אימות עם מספר אקראי המבוסס על חתימה דיגיטלית:

המאמת שולח למוכיח ערך אקראי חד פעמי במצב קריא וכל מה שהמוכיח עושה הוא חותם על ערך זה ומחזירו למאמת יחד עם החתימה. פרוטוקול אימות דו-צדדי מבוסס חתימה דיגיטלית, זהה לקודמו מלבד זאת ששני הצדדים שולחים זה לזה ערכים אקראיים חתומים בעזרת מנגנון החתימה הדיגיטלית שלהם.

Server Configuration

תמיכה בפרוטוקולים חזקים בלבד:

לפרוטוקולי SSL יש מספר רב של חולשות ומומלץ לא להשתמש בהם כלל. יישומי אינטרנט למטרות כלליות צריכים לתמוך רק ב-TLS 1.2 ו-TLS 1.3 כאשר כל הפרוטוקולים האחרים מושבתים. במקום שידוע כי שרת אינטרנט צריך לתמוך בלקוחות מדור קודם עם דפדפני אתרים שאינם נתמכים (כגון Internet Explorer 10), יתכן ויהיה צורך לאפשר ל-TLS 1.0 לספק תמיכה.

במקרה שלא ניתן למנוע שימוש בפרוטוקולי legacy יש להוסיף את ההרחבה **"TLS_FALLBACK_SCSV"** על מנת למנוע מתקפות downgrade מצד לקוח.

תמיכה בצפנים חזקים בלבד:

ישנם מספר רב של צפנים (ciphers בלעז) וחבילות צפנים (cipher suites) ש-TLS תומך בהם. רמת האבטחה מושפעת ישירות מהצופן שנבחר. כשמתאפשר, תמיד יש לשאוף לאפשר שימוש בצופן GCM (Galois Counter Mode). עם זאת, במידה ונדרש לספק תמיכה ל-legacy clients, צפנים אחרים עשויים להיות הכרחיים.

לכל הפחות, יש להשבית את הצפנים הבאים לצמיתות:

- Null ciphers
- Anonymous ciphers
- EXPORT ciphers

שימוש בפרמטרי Diffie-Hellman חזקים

כאשר משתמשים בצפנים המשתמשים בהחלפת מפתח מסוג Diffie Hellman (נוכל לראות אותם מסומנים בשם הצופן על ידי ראשי התיבות "DHE" או "EDH") יש להשתמש בפרמטרים של דיפי-הלמן מאובטחים באורך של לפחות 2048 ביט. נדגים כיצד ניתן לייצר מפתח מהסוג הנ"ל:

```
openssl dhparam 2048 -out dhparam2048.pem
```

נסתכל על תוכן המפתח כפי שלמדנו בפרק של חתימת תעודה דיגיטלית:

```
C:\Users\Jonathan Elkabas>openssl dhparam -in dhparam2048.pem -noout -text
    DH Parameters: (2048 bit)
        prime:
            00:cf:ec:ec:0b:44:17:d3:05:29:0e:c3:0e:03:2b:
            9b:61:2a:97:e7:57:66:e5:54:83:5f:54:b7:c0:6e:
            c6:19:9a:46:80:66:12:e4:06:b5:71:82:47:e1:2b:
            82:c8:b8:13:9d:be:87:cb:e6:f9:00:94:39:d0:4d:
            fe:ab:92:4b:19:62:87:ac:04:df:1c:e4:dd:56:5f:
            fb:55:53:d8:33:d1:a9:14:e2:b5:60:ea:4c:3d:21:
            46:b7:e2:a4:cf:f3:24:28:94:cd:fd:6b:51:22:3c:
            0b:95:73:36:64:8a:d6:2a:85:d1:1f:93:99:89:53:
            c9:40:c1:c6:15:6d:29:5e:13:60:4b:a0:48:e4:f3:
            94:ab:90:f6:32:9c:02:c6:6e:a2:d0:1c:f6:85:5d:
            ca:ae:40:2c:67:1d:90:cb:cd:d2:8d:81:48:81:6c:
            73:bd:34:50:fb:e1:37:3e:bc:9f:05:5d:31:46:29:
            12:e4:36:47:0d:19:af:5c:19:71:6e:71:4f:7e:d3:
```

```
        c4:ea:82:1c:c0:cd:2b:66:8f:0d:19:7a:d2:b5:30:
        a0:d7:a1:61:92:44:63:77:25:86:9a:7c:bc:ac:5e:
        16:29:f1:58:40:82:fb:e0:9e:38:b0:df:98:ba:de:
        22:56:f7:bd:1b:9a:1f:44:98:61:95:a6:f4:1f:d5:
        9c:db
    generator: 2 (0x2)
```

באתר WeakDH קיים מדריך איכותי כיצד לקנפג שרתים (כגון apache HTTP, ngix, Microsoft IIS, apache Tomcat וכו') בצורה נכונה עם מפתח DH:

https://weakdh.org/sysadmin.html

ביטול דחיסה

יש להשבית את דחיסת (compression) TLS על מנת להגן מפני פגיעות המכונה CRIME שעלולה לאפשר לשחזר מידע רגיש כגון Cookies הפעלה על ידי תוקף. טכניקה זו מאפשרת לתוקף לחטוף הפעלות SSL על ידי גניבת קובצי Cookies מאובטחים של משתמשים באמצעות הזרקת JavaScript לדפדפן.

patching ספריות קריפטוגרפיות

בנוסף לפגיעויות בפרוטוקולי SSL ו- TLS היו מספר רב של פגיעויות היסטוריות בספריות הממשות SSL ו-TLS כאשר Heartbleed (כתוצאה מפגיעות בספריית Openssl, הבאג אפשר לכל אחד [סטטיסטית] לקרוא את זיכרון המערכות המוגנות) הייתה הידועה ביותר. חשוב לוודא שהספריות הללו מתעדכנות ב-patching האבטחה האחרונים ביותר.

בדיקת קונפיגורציות השרת

לאחר שהשרת הוקשח, חשוב להריץ מספר בדיקות (tests) על מנת לוודא כי הוא אכן מקנופג ועומד בדרישות שדרשנו. ישנם מספר קלים שעוזרים לבצע את הבדיקות בצורה מהירה ואיכותית .

Online tools:

- SSL Labs Server Test
- Observatory by Mozilla

Offline tools:

- O-Saft - OWASP SSL advanced forensic tool
- testssl.sh - Testing any TLS/SSL encryption
- SSLScan - Fast SSL Scanner
- SSLyze

מימוש TLS באפליקציה

בכל הקשור למימוש עצמו של TLS ביישום עצמו, ישנם כמה כללי בסיס שמומלץ לעקוב אחריהם:

השמת TLS בכל הדפים

יש להשתמש ב-TLS לכל הדפים, ולא רק באלו שנחשבים רגישים כמו דפי login או Checkout. אם ישנם דפים שאינם אוכפים את השימוש ב- TLS הם עשויים לתת לתוקף הזדמנות לבצע sniffing למידע רגיש כגון session tokens, או להזריק JavaScript זדוני ל-responses של היישום לביצוע התקפות אחרות נגד המשתמש.

עבור יישומים ציבוריים, יתכן ויהיה נכון לקשר את שרת האינטרנט לחיבורי HTTP לא מוצפנים בפורט 80, ולאחר התממשקות ראשונית לנתב את החיבור ישר עם permanent redirect (HTTP 301) על מנת לספק חוויה טובה יותר למשתמשים המקלידים ידנית את שם הדומיין . יש לתמוך בזה באמצעות header של **HSTS** (HTTP Strict Transport Security) בכדי למנוע מאותם משתמשים גישה בעתיד לאתר דרך HTTP.

אי-ערבוב בין תוכן TLS ללא TLS

דף הזמין באמצעות TLS לא אמור לכלול משאבים (כגון JavaScript או CSS) קקבצים שנטענים באמצעות HTTP לא מוצפן. משאבים לא מוצפנים יכולים לאפשר לתוקף לבצע sniffing ל-session cookies באתר או לבצע הזרקת קוד לדף. כמו כן, חשוב לציין שדפדפנים מודרניים יחסמו ניסיונות לטעון תוכן פעיל דרך HTTPלא מוצפן לדפים מאובטחים.

השמת דגל cookie "Secure"

כל ה-cookies צריכות להיות מסומנות עם attribute "secure", שאומר לדפדפן לשלוח אותן רק דרך חיבור HTTPS מוצפן על מנת למנוע ניסיון של משתמש זדוני לעשות sniffing לחיבור HTTP לא מוצפן. הגדרה זו חשובה גם במקרה והאתר לא מאזין בפורט 80 (HTTP) מכיוון שתוקף יכול לבצע active MITM ולהציג למשתמש spoofed webserver בפורט 80 וכך לגנוב את מידע החיבור של המשתמש.

מניעת caching למידע רגיש

למרות ש- TLS מספק הגנה על נתונים בזמן שהם עוברים, הוא אינו מספק כל הגנה על נתונים ברגע שהם מגיעים למערכת שביקשה אותם. ככאלה, מידע זה עשוי להיות מאוחסן בזיכרון המטמון של דפדפן המשתמש, או על ידי כל intercepting proxies המוגדרים לביצוע פענוח TLS.

כאשר נתונים רגישים מוחזרים בתגובות, יש להשתמש ב-header של HTTP על מנת להורות לדפדפן ולשרתי פרוקסי לא לשמור במטמון את המידע, בכדי למנוע את אחסונו או להחזרתו למשתמשים אחרים. ניתן להשיג זאת על ידי הגדרת ה-headers של HTTP הבאים בתגובה:

```
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: 0
```

שימוש ב-HSTS

כפי שהוזכר מקודם, HTST מורה לדפדפן של המשתמש תמיד לבצע request לדף האתר דרך HTTPS, ובנוסף מונע ממשתמשים לעקוף את האזהרות בנוגע לתעודות לא חוקיות. על הטמעת HSTS אפשר לקרוא במסמך הבא:

HTTP Strict Transport Security cheatsheet

שימוש בתעודות בצד לקוח

בתצורה טיפוסית משתמשים ב-TLS עם תעודות בצד שרת על מנת שהלקוח יוכל לאמת את זהות השרת, ולספק חיבור מוצפן ביניהם. עם זאת, קיימות שתי חולשות עיקריות בגישה הזו:

- לשרת אין שום מנגנון לאימות זהות הלקוח (מה אם מדובר במערכת שמשאב העיקרי שלה מבחינתנו הוא דווקא השרת?)
- ניתן לבצע interception לחיבור על ידי תוקף שמסוגל להשיג תעודה שתקפה ל-domain.

שימוש בתעודות צד לקוח מטפל בסוגיות הללו על ידי דרישה שהלקוח יוכיח את זהותו לשרת עם תעודה משלו. זה לא רק מספק אימות חזק של זהותו של הלקוח, אלא גם מונע מגורם ביניים לבצע פענוח TLS גם אם הם סמכו על תעודת CA במערכת של הלקוח, אבל... זה קדר רציני.

לעתים רחוקות משתמשים בתעודות לקוח במערכות ציבוריות עקב מספר סיבות עיקריות:

- הנפקת וניהול תעודות לקוח גוררת תקורות (overheads) אדמיניסטרטיביות משמעותיות.
- משתמשים לא טכניים עשויים להיאבק להתקנת תעודות בצד לקוח.
- פענוח TLS המשמש ארגונים רבים יביא לכישלון באימות של תעודות הלקוח.

עם זאת, הפוטנציאל של תעודות צד לקוח ביישומים או APIs בעל ערך גבוה, במיוחד כאשר ישנם מספר קטן של משתמשים מתוחכמים מבחינה טכנית, או כאשר כל המשתמשים הם חלק מאותו ארגון (או מערכת / כלי).

נושא חשוב לא פחות מקינפוג השרת והאפליקציה להטמעת TLS בצורה נכונה הוא **מימוש נושא התעודות** בצורה טובה. למעשה זו הייתה מטרת העל של המסמך אך מכיוון ש-TLS הוא פרוטוקול שאנו נפגוש כמעט בכל פרויקט בתצורה כזו או אחרת, הוחלט להקדיש לו תת פרק שלם. עם זאת, מטרת התת פרק הבא - "*המלצות למימוש תעודות בצורה נכונה*" הינה לכנס ולסכם את כל המסמך למספר עמודים בודדים על מנת להוות אבן בניין ורפרנס איכותי בכל תהליך פרויקטלי עתידי.

המלצות למימוש תעודות בצורה נכונה

שימוש במפתחות חזקים ומוגנים

ראשית, חובה לציין שמפתח יחיד **חייב** לשמש לפעולה **אחת ויחידה**. כלומר, לאחר יצירת המפתח הפרטי, יש להשתמש בו למטרה אחת בלבד כדוגמת הצפנה, אימות זהות, חתימה דיגיטלית, key wrapping וכו'. שימוש מרובה באותו מפתח לתהליכים קריפטוגרפיים עתיד להחליש אותם ולחשוף אותם לפגיעויות נוספות (פחות או יותר אותו רעיון כמו הקטע של מחזור הסיסמא בכל רישום שכולם עושים - פויה).

עיקרון זה אינו מונע שימוש במפתח בודד במקרים בהם אותו תהליך מסוגל לספק מספר services. אלא מדובר על מקרים בהם למשל חתימה דיגיטלית מספקת authentication integrity ו- source authentication באמצעות חתימה דיגיטלית יחידה או כאשר ניתן להשתמש במפתח סימטרי יחיד להצפנת ואימות נתונים בפעולה קריפטוגרפית אחת (למשל, באמצעות פעולת authenticated-encryption לעומת פעולות הצפנה ואימות נפרדות זו מזו).

טבלה המסכמת את רמת האבטחה למצפנים סימטריים, אלגוריתמים אי-סימטריים ואורך מפתחות על פי NIST (עדכני למאי 2020):

ECC (ECDSA, EdDSA, DH, MQV)	IFC (RSA)	FFC (DSA, DH, MQV)	Symmetric Key Algorithms	Security Strength
f=160-223	k=1024	L=1024, N=160	2TDEA	**< 80**
f=224-255	k=2048	L=2048, N=224	3TDEA	**112**
f=256-383	k=3072	L=3072, N=256	AES-128	**128**
f=384-511	k=7680	L=7680, N=384	AES-192	**192**
f=512+	k=15360	L=15360, N=512	AES-256	**256**

מקרא:

Security Strength - רמת האבטחה המקסימאלית שמתאפשרת (בסיביות) כתלות באלגוריתם ובאורך המפתח. יש לציין כי רמת האבטחה היא לא בהכרח אורך המפתח עקב מתקפות על אלגוריתם אשר מאפשרות קיצור מספר פעולות החישוב הנדרשות לפיצוח.

Symmetric Key Algorithms - אלגוריתמים סימטריים המאפשרים את רמת האבטחה המקסימלית כפי שמצוינת בעמודה אחת. **2**TDEA, **3**TDEA הם TDEA (Triple DES) עם 2 מפתחות ו-3 מפתחות שונים. AES**XXX** מסמלים את מספר הסיביות של אורך המפתח באלגוריתם.

- **FFC** - גודל מינימאלי של הפרמטרים אשר מזוהים עם הסטנדרט של finite-field cryptography (FFC) כאשר **L** מציין את גודל המפתח הציבורי ו-**N** את אורך המפתח הפרטי.
- **IFC** - integer-factorization cryptography. k מציין את גודל המפתח ואת N modulus שהאלגוריתם מתבסס עליו.
- **ECC** - elliptic curve cryptography, f מציין את אורך המפתח לאלגוריתמים המבוססים על עקומות אליפטיות.

על פי **NIST**, אלגוריתמים \ צירופים של אורכי מפתחות בעלי ערך של **security strength < 112** אינם מורשים להגנה על קריפטוגרפיה על מידע ממשלתי פדרלי (למשל, הצפנת נתונים או הפקת חתימה דיגיטלית) ויש לחדול את השימוש בהם.

מתוך הבנה שכל מס' שנים שיטות פיצוח מתקדמות ומאפשרות יותר פעולות חישוביות בזמן נתון, המפתח הפרטי שבאמצעותו יוצרים את התעודה חייב להתאים בחוזקו לזמן החיים שמתוכנן לתעודה. כפי שמצוין ב-NIST, תחת שום נסיבות אין להשתמש במפתחות **RSA** באורך 1024. אורך הגודל המינימאלי הוגדר לשימוש הוא 2048 סיביות (2013). בנוסף, **אורך המפתח צריך להיבדק בכל חתימה בתעודה, כולל תעודת השורש.** אותן אזהרות חלות גם על מפתחות **DSA**, **ECC** בהתאם לטבלה.

יש להגן על המפתח הפרטי מפני גישה לא מורשית באמצעות הרשאות מערכת קבצים ושליטה טכנית ומנהלית כגון smart card או HSM (Hardware Security Module).

HSM שומר מפתחות דיגיטליים, מבצע פונקציות הצפנה ופיענוח לחתימות דיגיטליות, אימות ומטודות קריפטוגרפיות נוספות. בדרך כלל, מודולים של HSM מגיעים בצורה של plug-in card או התקן חיצוני המתחבר ישירות למחשב או לשרת רשת.

שימוש באלגוריתמי hash חזקים

חוזק פונקציית ה-hash שיעשה בה שימוש באלגוריתם או ביישום מעורך לפי אורך הפלט של הפונקציה. משפחות SHA2, SHA3 מכילות בשם פונקציית ה-hash את מס' הסיביות ביציאת הפונקציה, כך למשל, SHA-224 מכיל פלט של 224 סיביות.

Collision Resistance	Security Strength
SHA-0, SHA-1, MD5	**< 80**
SHA-224, SHA-512/224, SHA3-224	**112**
SHA-256, SHA-512/256, SHA3-256	**128**
SHA-384, SHA3-384	**192**
SHA-512, SHA3-512	**> 256**

לכל הפחות, יש להשתמש במשפחת SHA2 (SHA-224, SHA-256, SHA-384, and SHA-512) ובמידת האפשר לשדרג את השימוש למשפחת SHA3. **אין לעשות שימוש ב-SHA-1, MD5.**

מדיניות מסודרת לתעודות לא בטוחות \ תעודות שפג תוקפן

על כל ארגון אשר בוחר לממש את נושא התעודות בעצמו חשוב לתת את הדעת ולקבוע מדיניות מסודרת לכל:

- תעודות שפג תוקפן
- תעודות שידוע כי גורם שלישי השיג גישה אליהן
- תעודות שנגנזו
- תעודות עם מספר סידורי חופף
- תעודות זהות (same same but different)
- תעודות שלא עומדות בתקן הארגוני

על הארגון להחליט כיצד הוא מגיב לכל וריאציה של התעודות הנ"ל - האם יש להתריע על כל מקרה? האם קיים צורך תפעולי\ מבצעי שמחייב לקבל את התעודה ורק לאחר מכן לבדוק את מקורה? האם להעביר את התעודה ל-CRL? כיצד יש לממש את ה-CRL? איפה לאחסן אותו? האם הוא אבסולוטי? שאלות רבות צצות וחשוב לתת את הדעת על כולן בצורה מסודרת (ומתועדתתתת).

השמת שמות domain נכונים

שם הדומיין (או ה-subject) של התעודה חייב להתאים **בשלמותו** לשם של השרת שמציג את התעודה. בעבר זה היה מאוחסן בשדה commonName (CN) של התעודה אבל גרסאות עדכניות של דפדפנים (כדוגמת chrome) מתעלמים משדה ה-CN ודורשים שה-FQDN יהיו בשדה subjectAlternativeName (SAN).

FQDN - Fully Qualified Domain Name, ידוע גם ככתובת דומיין אבסולוטית המורכבת מ-hostname ו-domain. למשל - myhost.mydomain.com.

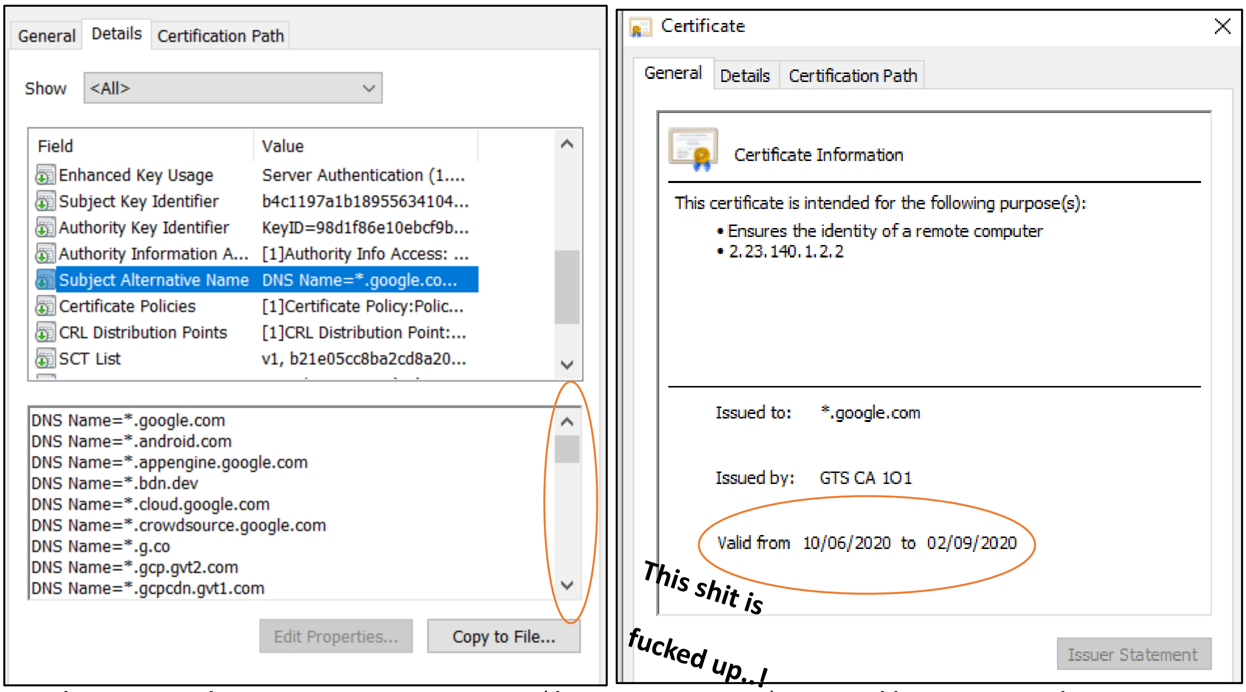
לשם תאימות, תעודות צריכות להכיל את ה-FQDN העיקרי ב-CN ואת הרשימה המלאה של FQDNs ב-SAN.

בנוסף, כאשר יוצרים תעודות יש לקחת מספר נושאים בחשבון:

- האם יש לכלול גם ה-subdomain של "www"
- לא לכלול non-qualified hostnames
- לא לכלול IP addresses
- לא לכלול שמות domain פנימיים על תעודות חיצוניות (הפונות החוצה)
- אם שרת כלשהו נגיש באמצעות FQDNs *פנימיים וחיצוניים* יש לקנפג אותו עם מספר תעודות (ולא אחת בלבד לשני המקרים).

שימוש זהיר ב-wildcard בתעודות

שימוש ב-wildcard (כוכבית - "*") מאוד נוח מכיוון שהוא מאפשר לכלול מספר subdomains בתעודה אחת של domain (כגון *.he.wikipedia.org). כתוצאה מהאפשרות של למתן תעודה אחת למספר רב של hosts, הרבה ארגונים עושים שימוש בתו כללי. קחו למשל את התעודה של Youtube:



היתרונות של שימוש בתו כללי ברורים (מאופן כספי ותפעולי) אך עם זאת שימוש בו עשוי לגרור הפרה לא מכוונת של הרשאות. כלומר, למשל במצב בו administrative privileges ניתנים לשרת תווך שבסה"כ צריך להתחבר לרשת ולכתוב איזשהו log למסד נתונים.

שנית, כאשר מספר מערכות חולקות תעודות תווים כלליים, הסבירות שהמפתח הפרטי לתעודה נפגע עולה, מכיוון שהמפתח עשוי להיות מוטמע מערכות מרובות. בנוסף הערך של המפתח הזה עולה משמעותית, מה שהופך אותו למטרה אטרקטיבית יותר עבור תוקפים פוטנציאלים. התעודה של גוגל שלמעלה **חותמת מעל 40 domains שונים**, תחשבו כמה ערך כלכלי יש למפתח היחיד הזה, הוא בוודאי חותם מאות שרתים.

כאשר מבצעים ניהול סיכונים לשימוש בתעודות עם תווים כללים יש לקחת בחשבון את הנקודות הבאות:

- יש לעשות שימוש בתעודות תווים כלליים היכן שבאמת קיים צורך חי ובוער, **לא לשם נוחות** ותו לא - **יש להימנע ככול הניתן משימוש בתעודות עם תווים כלליים.**
- יש לשקול את השימוש ב-ACME (Automated Certificate Management Environment - אינטראקציות אוטומטיות בין CAs ל-web services של המשתמשים) על מנת לאפשר למערכות לבקש חידוש לתעודות שברשותן בצורה אוטומטית במקום שימוש בתו כללי.
- במידה ועושים שימוש בתעודות תו כללי, יש לודא כי מיישמים את <u>**כל**</u> התכונות לביטול תעודות (CRL וכו') העומדות לרשות המערכת.

- **אסור** לעשות שימוש בתעודת תו כללי למערכות עם trust level שונה.
  - 2 עמדות VPN Gateways יכולות לחלוק תעודת תו כללי
  - מספר instances של אפליקציית web יכולים לחלוק תעודה
  - VPN Gateway ושרת רשת פומבי **לא צריכים** לחלוק תעודות תו כללי
  - שרת רשת פומבי ושרת פנימי **לא צריכים** לחלוק תעודות תו כללי
- יש לשקול שימוש בשרת reverse proxy אשר מבצע TLS termination, כך תווי מפתחות כלליים יהיו מוגבלים רק למערכת אחת.
- יש לתחזק רשימה של כל המערכות החולקות תעודה על מנת לאפשר עדכון של התעודה כאשר פג התוקף או כאשר המפתח של התעודה נחשף.
- להגביל את מרחב השימוש בתעודות תו כללי ל-subdomain (כדוגמת *.foo.example.org).
- במידה ומחליטים לעשות שימוש בתעודות תו כללי יש לנסות ככול הניתן להגביל את השימוש בהן רק ל-lowest level domain והימנע ככול הניתן משימוש ב-root level domains.

אבל רק קחו בחשבון ש... גוגל שמים על כל זה פס אז לא יודע כמה הייתי נלחם על זה מול מנהלים ואנשי IT. אבל מצד שני אם אין לכם את התקציב שיש לגוגל להגן על המפתח שלה אז...

השמת CA מתאים על בסיס הצורך של משתמש היישום

על מנת שמשתמשים יסמכו על התעודה היא חייבת להיות חתומה על ידי CA מוכר. עבור יישומים הפונים לאינטרנט, זה חייב להיות אחד מה-CA המוכרים ואמינים באופן אוטומטי על ידי מערכות הפעלה ודפדפנים.

למערכות \ אפליקציות פנימיות ניתן להשתמש ב-CA פנימי. משמעות הדבר היא שה-FQDN של התעודה **לא ייחשף** (לא לרשות CA חיצונית או ברשימות שקיפות תעודות בפומבי). עם זאת, משתמשים יסתמכו על התעודה רק לאחר שיבצעו import ויסתמכו על תעודת ה-CA הפנימי ששימש לחתימה עליהם.

<u>יש לוודא כי העברת\ צריבת התעודה נעשית בצורה מאובטחת ובטוחה</u>. (אל תרוצו לי עם אותו / SanDisk USB שהזמנתם באלי-אקספרס על 100+ מחשבים שונים ותקוו לטוב).

שימוש ברשומות CAA על מנת לאכוף אלו CAs יכולים להנפיק תעודות

Crtification Authority Authorization (CAA) לרשומות DNS יכול לשמש להגדרת אילו CAs יכולים להנפיק תעודות לדומיין. רשומות CAA מכילים tag, value ו-flag אשר מציין את חשיבות הרשומה. למשל, על מנת לציין ש- ca.example.net הוא ה-CA היחיד שמורשה להנפיק תעודות עבור example.com ולכל subdomains שלו, נשתמש ברשומת CAA:

```
example.com.  IN  CAA 0 issue "ca.example.net"
```

כדי לא לאשר שום הנפקת תעודות עבור example.com, נאפשר את ההנפקה רק לרשימת מנפיקים ריקה:

example.com. IN CAA 0 issue ";"

ישנם עוד פיצ'רים רבים ל-CAA אך הם מעבר להיקף של המסמך הזה. לקריאה מעמיקה יותר אפשר לפנות אל RFC 6844.

השימוש במנגנון הזה יכול לעזור למנוע מתוקף לקבל תעודות לא מורשות עבור דומיין באמצעות CA פחות מהימן. כאשר מיישמים את זה על כל ה-subdomains, זה יכול לספק גם יתרון מנהלתי על ידי הגבלת באילו CAs מנהלי רשת \ מפתחים יכולים להשתמש ובאמצעות זה למנוע מהם שימוש לא מאושר בתעודות תו כללי.

תמיד להעביר את כל התעודות הנדרשות

על מנת לאמת את האותנטיות של התעודות, על המערכת\ הדפדפן של המשתמש לבחון את התעודה ששימשה כדי לחתום עליו ולהשוות אותה לרשימת ה-CAs שאצלו במערכת. במקרים רבים האישור אינו חתום ישירות על ידי ה -root CA, אלא במקום זאת הוא חתום על ידי intermediate CA ועליו חותם ה-root CA.

במידה ושרשרת האימון לא קונפגה כראוי\ לא סיפקו את כלל התעודות למערכת, משתמש שאינו מכיר או סומך על ה-intermediate CA יפיל את אימות התעודה, גם אם המשתמש בוטח ב-root CA הסופי מכיוון שהוא לא יכול ליצור את שרשרת אמון בין התעודה שמוצגת לו לתעודת השורש. על מנת להימנע מכך, מומלץ לספק למערכת את כל תעודות הביניים של ה-intermediate CAs.

"The application serving the certificate has to send the complete chain, this means the server certificate itself and all the intermediates."

בדיקת שרשרת התעודות צריכה להתבצע בצורה הבאה: התעודה של השולח צריכה להיות במיקום האחרון ברשימה. לאחר מכן (כאשר אנחנו סופרים מהסוף להתחלה), יש לבדוק כי כל תעודה ברשימה אישרה את התעודה העוקבת אליה. מכיוון שמנגנון אימות התעודות (certificate validation) דורש שמפתחות השורש יחולקו בצורה נפרדת (ויוחסנו בצד לקוח), התעודה שחתומה על עצמה (self-signed) ומכירה בתעודת השורש **יכולה** להיות מושמטת מתהליך הבדיקה (תחת ההנחה שמכיוון שלצד לקוח כבר יש את התעודה הוא יכול לאשר אותה במקרה הצורך).

RFC-3280 מתמצת את עיקרון האלגוריתם בצורה טובה (פסאודו קוד):

```
To meet this goal, the path validation process verifies, among other
things, that a prospective certification path (a sequence of n
certificates) satisfies the following conditions:

   (a)  for all x in {1, ..., n-1}, the subject of certificate x is
   the issuer of certificate x+1;

   (b)  certificate 1 is issued by the trust anchor;

   (c)  certificate n is the certificate to be validated; and

   (d)  for all x in {1, ..., n}, the certificate was valid at the
   time in question.
```

בנוסף, חשוב לציין כי במידה ובוחרים לממש את המנגנון שבודק את שרשרת התעודות (Certificate Validation), **אסור לממש אותו באמצעות רקורסיה.** לא רשום סטנדרט\ תקן למספר מקסימאלי של שרשורי תעודות שאפשר לבצע (ברמת העיקרון יכולים להיות עשרות intermediate CAs). מכיוון שרקורסיה איטית ומעמיסה על הזיכרון (כל קריאה רקורסיבית מוסיפה עצמה למחסנית ביחד עם המשתנים הרגעיים שברשותה), יכול להיווצר DOS מניסיון הקריאה של תעודת הקצה בצורה רקורסיבית. תוקף גם יכול לנצל חולשה זו ולשלוח במכוון שרשור תעודות ארוך מאוד.

אי לכך, **חובה** לעשות שימוש בשדה pathLenConstraint בתעודות X.509v3 אשר מגדיר במפורש את אורך שרשור האפשרי לתעודות ל-CA.

שימוש בשדות הרחבה נכונים

שדות X.509v2 - אין צורך ולא מומלץ לעשות בהם שימוש: Issuer Unique Identifier, Subject Unique Identifier. פירוט על נחיצות שדות X.509v3:

Extension	Mandatory or Optional
Authority Key Identifier(2.5.29.19)	Appears in all certificates
Basic Constraints(2.5.29.35)	Appears in all CA certificates
Certificate Policies(2.5.29.32)	Appears in all certificates
CRL Distribution Points(2.5.29.31)	Optional. Appears in all certificates whose revocation status is distributed by indirect CRLs or segmented CRLs.
Enhanced Key Usage(2.5.29.46)	
Issuer Alternative Name(2.5.29.8)	
Key Usage(2.5.29.15)	Appears in all certificates
Name Constraints(2.5.29.30)	Appears optionally in CA certificates
Policy Constraints(2.5.29.36)	Appears optionally in CA certificates.
Policy Mappings(2.5.29.33)	Appears only in CA certificates. Appears only when the issuer and subject issue certificates under different policies
Private Key Usage Period(2.5.29.16)	

Subject Alternative Name(2.5.29.17)	**Optional, but commonly used**
Subject Directory Attributes(2.5.29.9)	
Subject Key Identifier(2.5.29.14)	**Appears in all CA certificates. May optionally appear in end entity certificates.**

סיכום

נסיים עם השאלה שהתחלנו איתה - למה לעזאזל אנשים סומכים על אתרים כמו PayPal ו-AliExpress ועל זה שתוקף לא יכול להאזין לתעבורה שיוצאת מהם ולחלץ את כל פרטי האשראי שלהם?

איך אני יודע שאני מתחבר ל-aliexpress.com המקורי ולא לחיקוי שלו?

כשהדפדפן שלי מתחבר לדומיין של אלי-אקספרס, הם עושים טקס קטן שנקרא **TLS handshake** שבמסגרתו אלי-אקספרס והדפדפן מסכימים על שפה משותפת - **סוג המפתח** (RSA/DSA/DH), **שיטת ההצפנה** (AES/TDES) **ושיטת הגיבוב** (SHA כלשהו) בהם הם ישתמשו לצורך ההצפנה בהמשך.

בהמשך הטקס, לאחר שהם רואים שהם אכן יכולים לדבר את אותה השפה, השרת (אלי-אקספרס) שולח את המפתח הציבורי שלו בתעודה חתומה על ידי CA לדפדפן. מערכת ההפעלה שלי מבצעת השוואה פשוטה ומגיעה למסקנה כי היא **מכירה וסומכת** על אותו CA מכיוון שהוא צרוב לה כ-root CA-ב certmgr.msc. כלומר, אם DigiCert (או כל חברה גדולה שחותמת על תעודות) מוכנה להיות ערבה לזהותו של aliexpress.com אין סיבה שאני לא אאמין שהוא אכן מי שהוא טוען שהוא. (מדגיש - עד פה אין שום דבר מתוחכם מלבד **אמון**).

בשלב הבא, הדפדפן שלנו מייצר ג'יבריש אקראי (מספר גדול) ומצפין אותו **בעזרת המפתח הציבורי של השרת עצמו** ושולח לשרת. השרת מנגד, שמחזיק במפתח הפרטי, מפענח את ההצפנה של המפתח הציבורי שלו וכעת גם לו וגם לדפדפן יש ג'יבריש aka סוד משותף - **מפתח סימטרי** זהה. הם 2 המכונות היחידות בכל האינטרנט שיודעים מה המפתח הסודי הזה.

זהו, מעכשיו כל data שנעביר ב-HTTP Session עם השרת יהיה מוצפן עם המפתח הזה מבלי לחשוש שמישהו יהיה מסוגל לקרוא את הפאקטות והתעבורה.

וזהו ☺. אם יש לכם שאלות ו\או הערות ו\או memes טובים, אשמח שתשלחו לי במייל:

jonathanelkabas@gmail.com

על הכותב

יהונתן אלקבס, בן 25, חוקר אבטחת מידע בחברה לא קטנה ולא פרטית. חובב קקטוסים, סודה ומכור לקפה.

מקורות

RFC-5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile https://tools.ietf.org/html/rfc5280

RFC-3739: Internet X.509 Public Key Infrastructure: Qualified Certificates Profile https://tools.ietf.org/html/rfc3739

RFC-4158: Internet X.509 Public Key Infrastructure: Certification Path Building https://tools.ietf.org/html/rfc4158

RFC-4210: Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP) https://tools.ietf.org/html/rfc4210

RFC-5077: Transport Layer Security (TLS) Session Resumption without Server-Side State https://tools.ietf.org/html/rfc5077

RFC-6844: DNS Certification Authority Authorization (CAA) Resource Record https://tools.ietf.org/html/rfc6844

RFC-4262: X.509 Certificate Extension for Secure/Multipurpose Internet Mail Extensions (S/MIME) Capabilities https://tools.ietf.org/html/rfc4262

RFC-2119: Key words for use in RFCs to Indicate Requirement Levels https://tools.ietf.org/html/rfc2119

SSL & Trasport Layer Security Protocol מאת אפיק קסטיאל:
https://www.digitalwhisper.co.il/files/Zines/0x02/DW2-1-SSL.pdf

TLS - חלק א' (הצפנות א-סימטריות, RSA הבסיס המתמטי) מאת שחר קורוט:
https://www.digitalwhisper.co.il/files/Zines/0x55/DW85-3-TLS-Part1.pdf

HTTPS למתקדמים: אימות זהות מאת ליאור בר-און:
https://www.geektime.co.il/https-part-2-identification/

A Security Flaw in the X.509 Standard https://csrc.nist.gov/csrc/media/publications/conference-paper/1996/10/22/proceedings-of-the-19th-nissc-1996/documents/paper075/paper.pdf

Key Management and Certificates https://www.cs.auckland.ac.nz/~pgut001/tutorial/T2_Key_Management.pdf

PKI: The OSI of a new generation https://www.cs.auckland.ac.nz/~pgut001/tutorial/T2a_X509_Certs.pdf

Fixing X.509 Certificates https://tersesystems.com/blog/2014/03/20/fixing-x509-certificates/

Fixing Certificate Revocation

https://tersesystems.com/blog/2014/03/22/fixing-certificate-revocation/

Get your certificate chain right

https://medium.com/@superseb/get-your-certificate-chain-right-4b117a9c0fce

Overview of Red Hat Certificate System Subsystems

https://access.redhat.com/documentation/en-us/red_hat_certificate_system/9/html/administration_guide/setting_up_certificate_system

Create Certificate Authority and sign a certificate with Root CA

https://www.golinuxcloud.com/create-certificate-authority-root-ca-linux/

Global OID reference database

https://oidref.com/