SSL & Trasport Layer Security Protocol

מאת אפיק קסטיאל (cp77fk4r)

הקדמה (Secure Sockets Layer & Trasport Layer Security Protocol)

פרוטוקול ה-SSL (קיצור של Secure Sockets Layer) פותח בתחילת שנת 1994 ע"י חברת NETSCAPE בכדי להוסיף נדבך לאבטחת רשת האינטרנט. במקור הפרוטוקול פותח בכדי להוסיף אבטחה לאפליקציות WEB, אבל כיום הוא יכול לשמש גם כמאפיין אבטחה עיקרי בחיבורים רבים אחרים, כגון NNTP, SMTP, או FTP (כמו למשל חיבור ה-JSCAPE לשרתי AIX) ועוד.

כמו שאפשר להבין משמו - הפרוטוקול מהווה שכבה נוספת מתחת לפרוטוקול הקיים, כדוגמת השימוש בו על גבי פרוטוקול ה-HTTP, או מה שאנחנו מכירים כיום יותר כ-"HTTPS", פרוטוקול ה-SSL "מתיישב" על גבי פרוטוקול ה-HTTP ודואג להצפנת הנתונים העוברים דרך הפרוטוקול התחתון.

ההצפנה בפרוטוקול ה-SSL עוצבה באופן כזה שהיא פועלת ברמת ה-Socket, ולכן השימוש הנכון בו הוא שימוש על גבי פרוטוקול אמין כגון TCP, בשל העיצוב שלו הוא אינו תלוי מערכת, מה שנותן לו את העדיפות כשמדובר במקרים שבהם צריכים להעביר מידע על גבי מספר פלטפורמות (כגון שימוש באינטרנט).

פרוטוקול ה-TLS (קיצור של Transport Layer Security) מבוסס ברובו על הגירסה השלישית של פרוטוקול ה-SSL, שיצאה כשנתיים לאחר הגירסה הראשונה של הפרוטוקול המקורי - בשנת 1996, שלוש שנים לאחר שיצאה הגירסה השלישית הושלם הפיתוח של ה-TLS ע"י החבר'ה מ-IETF (קיצור של Internet Engineering Task Force) והוא הובא לשימוש ציבורי (בתחילה של חברות אשראי בעיקר) לקראת סוף שנת 1999.

הרעיון הכללי

הרעיון מאחורי שיטת העברת המפתחות של הפרוטוקול מאוד מזכירה את ה-PGP ואת השימוש במפתח ציבורי של אחד מהגורמים (הצד המארח), אך החידוש המרכזי כאן הוא שימוש בגורם חיצוני - צד שלישי אמין - בכדי לאמת את נתוני ההצפנה לפני השימוש הראשוני בהם.

השימוש בהצפנה אסימטרית חזק בהרבה משימוש בהצפנה סימטרית, אך שימוש בה בפרוטוקול ה-FTP או ה-HTTP, למשל, פחות יעיל בהשימוש בהצפנה סימטרית, מפני שהוא יפגום מאוד ב"חווית הגלישה" – המידע יעבור באופן איטי יותר, ולכן השימוש בו לאורך כל התקשורת - נכון לימים אלה - לא מתקבל על הדעת.

לחיצת היד של הפרוטוקול משתמשת בשתי השיטות - היא לוקחת את חוסנה של ההצפנה האסימטרית מצד אחד, ומצד שני היא לוקחת את יעילותה וגמישותה של ההצפנה הסימטרית. הרעיון הוא שאם התקשורת נפלה קורבן למתקפת Men In The Middle מוצלחת ואנחנו נבצע את החלפת מפתחות ההצפנה באופן פשוט כל המנגנון הלך, הצד השלישי מחזיק במפתח גם הוא, הוא יוכל לקבל את המידע המוצפן, לפענח אותו, לערוך אותו או לדלות ממנו את המידע הרגיש (סיסמאות, מספרי כרטיסי אשראי וכו') ולשחרר אותו ליעדו (במקרה כזה גם לא תהיה שום אפקטיביות למנגנוני Time-Stamp למניהם, מפני שלא תהיה בעיה לתוקף לזייף אותם ב-Request/Response הנוכחי). אך במקרה והתוקף מקבל מידע מוצפן בעזרת מפתח שאין לו - הוא לא יוכל לבצע בעזרתו שום דבר- אם הוא ינסה לפגום במידע, להחליפו באחר או לבצע כל מניפולציה, ולו הקטנה ביותר, בזמן שהלקוח ינסה לבצע פיענוח על החבילה המתקבלת ויתקל בבעיות, הוא יבין שגורם שלישי "נגע" בחבילה, יזרוק אותה מיד וידע לא לסמוך על התקשורת הנוכחית.

פרוטוקול ה-SSL, תומך במספר אלגוריתמים לשני שלבי ההצפנה שבהם הוא משתמש.

- **להצפנה ה-אסימטרית (שלב שני - שלב החלפת המפתחות):**
  הפרוטוקול תומך ב-RSA, Diffie-Hellman , DSA ובגירסתו השלישית של הפרוטוקול נוספה היכולת להשתמש אף ב-FORTEZZA.

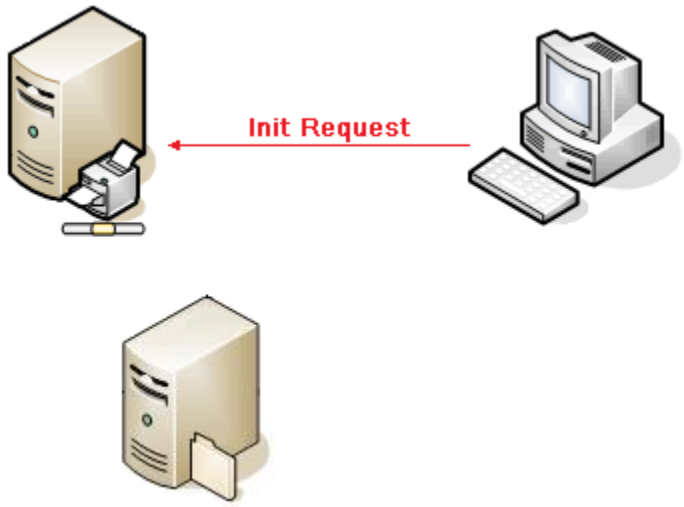
- **להצפנה הסימטרית (שלב שישי - העברת המידע הרגיש):**
  הפרוטוקול תומך ב-AES, משפחת ה-RC (הנפוצים ביותר לשימוש הם ה-RC2 וה-RC4), IDEA, וכמובן ב-DES וב-TripleDES.

- במקרים רבים אפשר למצוא נדבך אבטחה נוסף והוא העברת המידע באופן מגובב. ברוב המקומות הסטנדרטיים מבוצע שימוש באלגוריתמי גיבוב מוכרים כגון MD5 (בגירסאות ישנות של הפרוטוקול ישנו שימוש אף ב-MD2 ו-MD4) ומשפחת SHA.

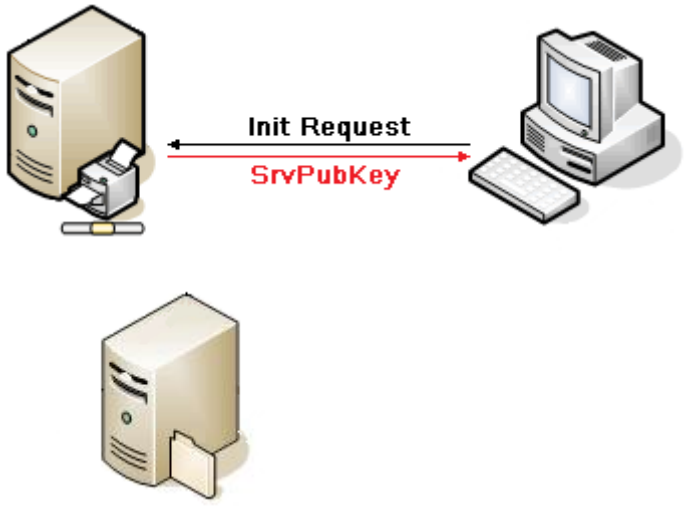
מנגנון ה-Handshake

מהלך לחיצת היד מתנהלת לפי האופן הבא:

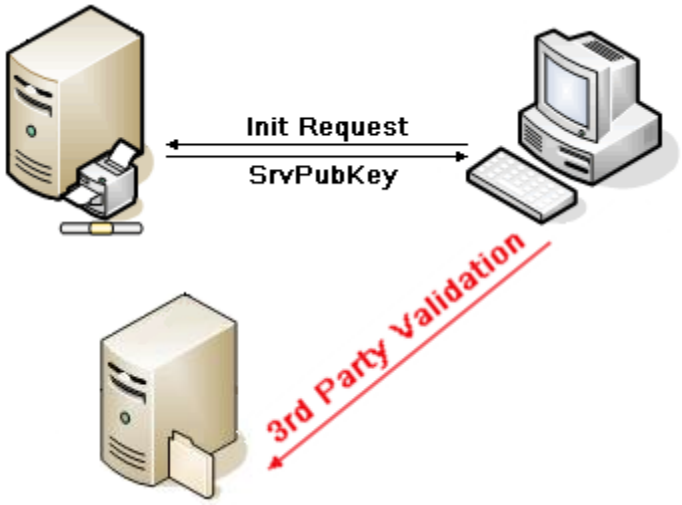
- **שלב ראשון -** הלקוח שולח בקשה ליצירת קשר עם השרת.



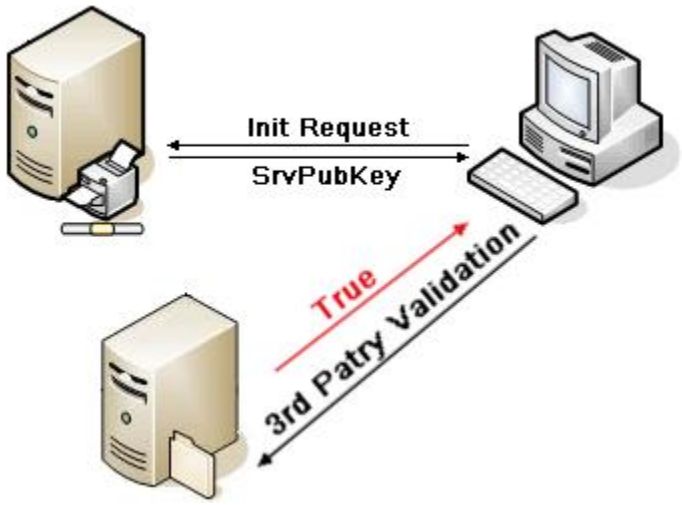
- **שלב שני -** השרת שולח Certificate ייחודי לו, אשר מכיל בין היתר את המפתח הציבורי שלו:



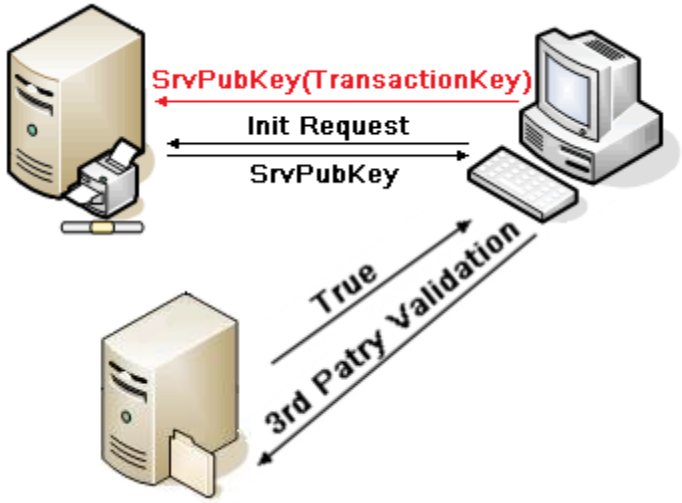
- **שלב שלישי -** הלקוח מתשאל צד שלישי (החברה המנפיקה של אותו ה-Certificate) ומוודא כי הוא אכן אותנטי.



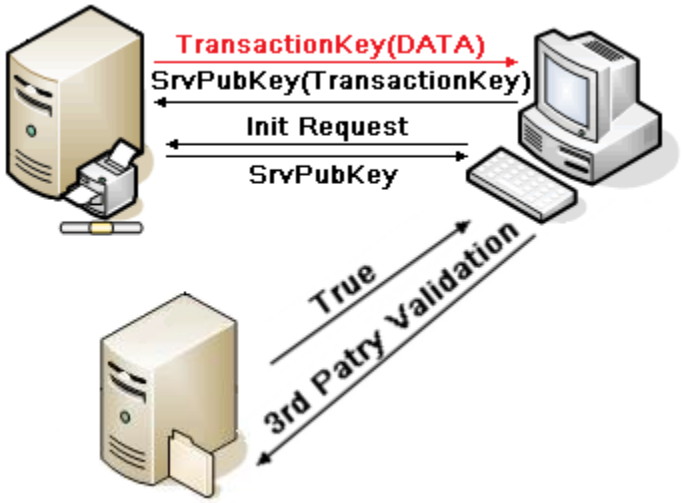
- **שלב רביעי -** הלקוח מקבל תשובה חיובית (במידה ואכן ה-Certificate אמיתי) מהגורם השלישי.



- **שלב חמישי -** הלקוח מצפין מספר רנדומאלי (Transaction Key) בעזרת המפתח הציבורי שאותו הוא שלף מתוך ה-Certificate המאושר ושולח אותו לשרת.



- **שלב שישי -** השרת מפענח את ההודעה שהגיע מהלקוח בעזרת המפתח הפרטי שברשותו, שולף מתוכה את המספר הרנדומאלי שאותו יצר הלקוח, ומצפין בעזרתו את המידע הרגיש שאותו הלקוח ביקש.



המפתח הרנדומאלי הזה מהווה את מפתח ההצפנה של השיחה, ומכאן- שכבת האבטחה תצפין בעזרתו כל חבילת מידע שתעבור דרכה ותדאג לפיענוחה (בעזרת אותו מספר) כאשר החבילה הגיעה ליעדה.

שימו לב שגם אם תוקף מאזין לתקשורת כבר מהשלב הראשון אין לו אפשרות לגלות מה מכילה חבילת מידע האחרונה שנשלחה בשלב השישי- המידע הרגיש שאותו ביקש הלקוח.

למה? כי בכדי להשיג את ה-Transaction Key על התוקף לפענח את המידע שנשלח ע"י הלקוח - מידע זה הוצפן באמצעות המפתח הציבורי של השרת ורק בעזרת המפתח הפרטי שלו יהיה ניתן לפענח את חבילת המידע הזו.

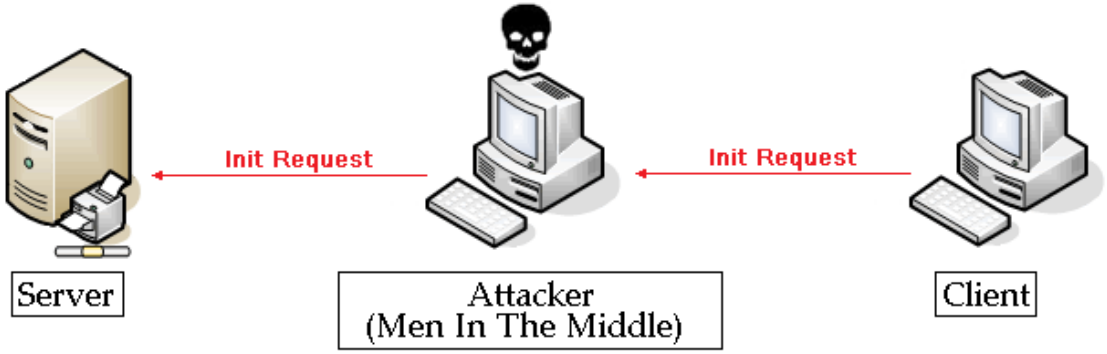
כמובן שהמנגנון מכיל רכיבי Time-Stamp הנשלחים מהלקוח לשרת המוצפנים בעזרת המפתח הציבורי של השרת בכדי למנוע מתוקף שהצליח ליירט את התקשורת בין שני המחשבים לבצע מתקפות Replay Attack או לבצע סילוף של חבילות המידע.

מנגנון ה-Certificate Authority

כמו שראינו בפרק הקודם, בשלב השני והשלישי ישנו שימוש ב-Certificate של ה-SSL, בשלב השני- השרת שולח ללקוח את ה-SSL Certificate שלו - שכולל את מפתחו הציבורי שבו הלקוח ישתמש בכדי להצפין את מפתח השיחה העתידית, ובשלב השלישי הלקוח שולח לגורם שלישי **מוגדר מראש** את פרטי השרת בכדי לאמת כי אכן השרת הוא מי שהוא טוען לזהותו, או בשפה המקצועית, כאן מתבצעת פעולת ה-CA (קיצור של "Certificate Authority").

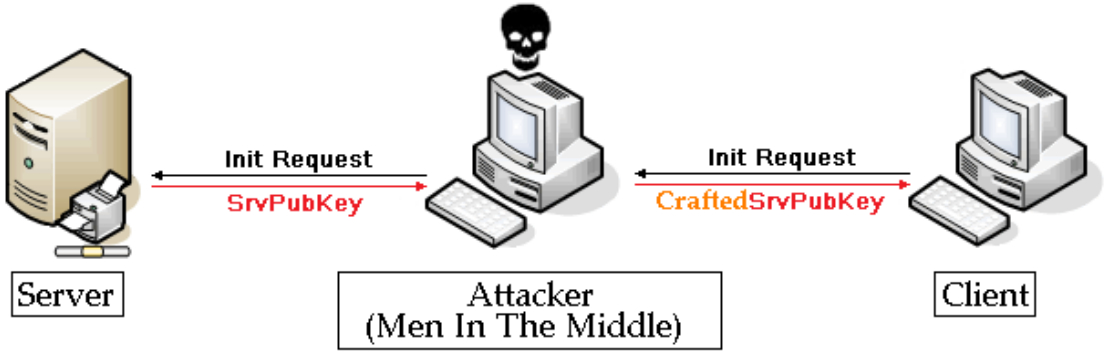
במקרים ואין שימוש ב-CA ובוצעה האזנה לאותו קו תקשורת שבו מתבצעת לחיצת היד שלנו, תוקף יוכל לנסות לבצע מניפולציה של המפתח הציבורי של השרת בכדי לגרום ללקוח להצפין את המידע הרגיש בעזרת מפתח ציבורי השייך לתוקף- וכך בעת השליחה לגנוב את המידע ולפענחו בעזרת המפתח הפרטי שלו עצמו, כמו בדוגמא הבאה:

- **שלב ראשון -** הלקוח שולח בקשת יצירת קשר עם השרת [על-גבי תקשורת תחת ציתות]:



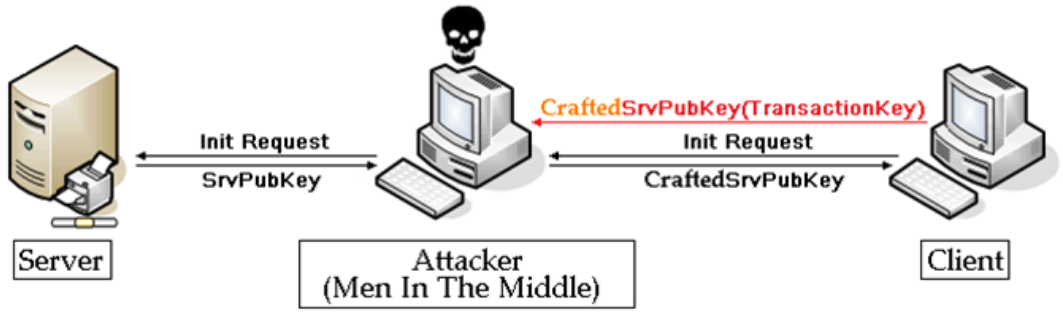
(הלקוח שולח את הבקשה לשרת > הבקשה מגיעה לתוקף > התוקף מעביר אותה לשרת מבלי לגעת בה)

- **שלב שני -** השרת שולח Certificate ייחודי לו אשר מכיל בין היתר את המפתח הציבורי שלו:



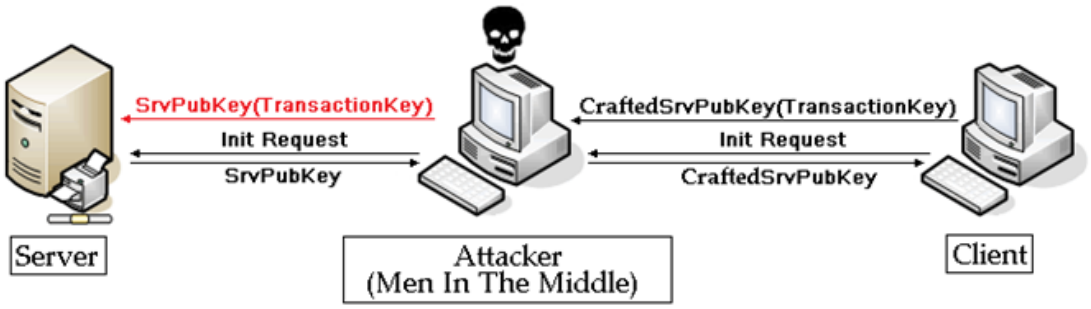
(השרת שולח תגובה המכילה את ה-Certificate, המכיל את המפתח הציבורי שלו ללקוח > התגובה מגיעה לתוקף > **התוקף עורך את פרטי המפתח הציבורי של השרת ומחליף אותו בפרטי המפתח הציבורי שלו (של התוקף)** > התוקף מעביר את תגובת השרת ללקוח).

- **שלב שלישי-** הלקוח מצפין מספר רנדומאלי (Transaction Key) בעזרת המפתח הציבורי שאותו הוא שלף מתוך ה-Certificate שקיבל מהשרת ושולח אותו לשרת.

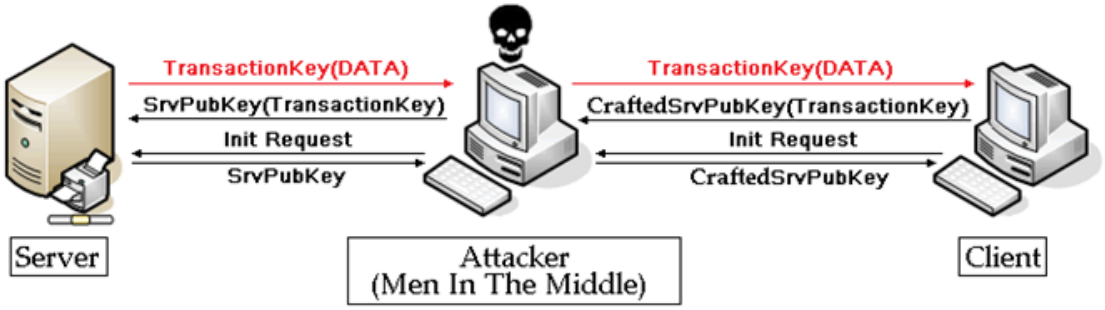


(הלקוח שולח לשרת מספר רנדומאלי בעזרת מפתח ההצפנה שקיבל מהשרת, לאחר שהתוקף ערך אותו > המידע מגיע לתוקף > התוקף מפענח את ההודעה ורואה מה המספר שנבחר ע"י הלקוח)

- **שלב רביעי -** התוקף מצפין את המספר הרנדומלי שקיבל מהלקוח בעזרת מפתחו הציבורי של השרת ושולח את החבילה לשרת:



- **שלב חמישי -** השרת מפענח את ההודעה שהגיע מהלקוח בעזרת המפתח הפרטי שברשותו, שולף מתוכה את המספר הרנדומאלי שאותו יצר הלקוח, ומצפין בעזרתו את המידע הרגיש שאותו הלקוח ביקש- ושולח ללקוח.



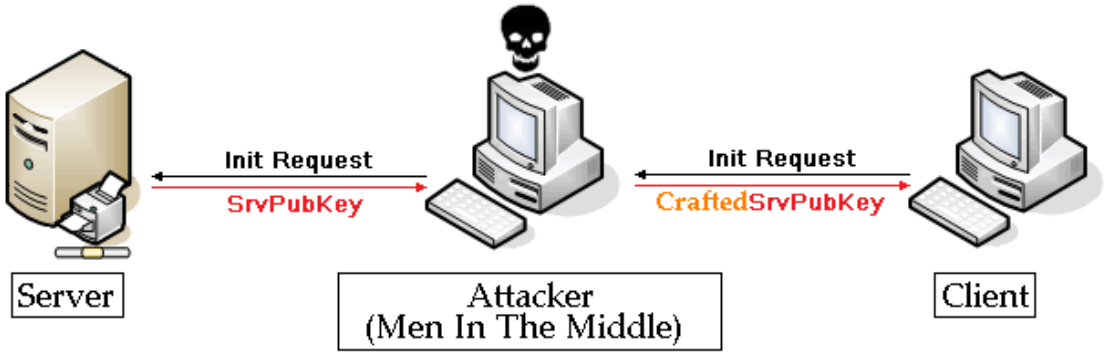
(השרת מצפין את המידע הרגיש שאותו ביקש הלקוח בעזרת מפתח ההצפנה שיצר הלקוח > השרת שולח את המידע המוצפן ללקוח > המידע מגיע לתוקף, התוקף מפענח את המידע, קורא אותו/ עורך אותו > התוקף מעביר את המידע ללקוח > GAME-OVER).

כמו שראינו, גם במקרה שיש שימוש בפרוטוקולי הצפנה והמידע נשלח באופן מוצפן, כל עוד אין לנו מנגנון שמאמת כי אכן אנחנו מדברים עם השרת האמיתי ולא עם מתחזה- המידע שלנו נתון לסכנה- גם במקרים כמו קניה באתרי אינטרנט המצב דומה:

- המידע המוצפן הגיע ללקוח (נניח- טופס תשלום הקנייה).
- הלקוח מכניס את כרטיס האשראי שלו ושולח לשרת.
- הפרוטוקול מצפין את המידע בעזרת אותו TransactionKey.
- המידע מגיע לתוקף- התוקף יודע כבר יודע מהו ה-TransactionKey, מפענח את החבילה- שולף את המידע הרגיש (לדוגמא- מספר האשראי), מצפין מחדש ושולח לשרת.

בדיוק בגלל התרחיש הנ"ל פותח מנגנון ה-Certificate Authority שנועד לאמת כי אכן השלב השני (שליחת המפתח הציבורי) בלחיצת היד מתבצע כמו שצריך, מבלי גורמים נוספים. השימוש הכללי של ה-CA הוא לאמת כי אותו מפתח ציבורי שקיבלנו מהשרת הוא אכן שייך לאותו שרת, וכי אין שום חשש להצפין בעזרתו את המידע הרגיש שאותו נרצה לשלוח לשרת, שימו לב- נחזור לשלב השני בתקשורת הנמצאת תחת האזנה:

- **שלב שני -** השרת שולח Certificate ייחודי לו אשר מכיל בין היתר את המפתח הציבורי שלו:



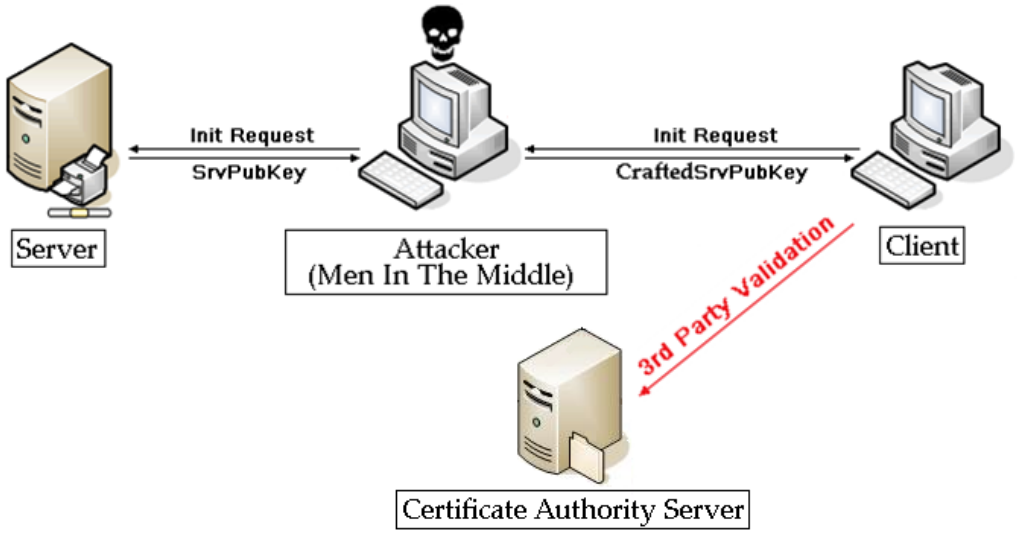
(השרת שולח תגובה המכילה את ה-Certificate המכיל את המפתח הציבורי שלו ללקוח > התגובה מגיעה לתוקף > **התוקף עורך את פרטי המפתח הציבורי של השרת ומחליף אותו בפרטי המפתח הציבורי שלו (של התוקף)** > התוקף מעביר את תגובת השרת ללקוח).

[כאשר מדובר בפרוטוקול התומך ב-"Certificate Authority" השלב הבא לא יהיה הצפנת מפתח השיחה ע"י הלקוח בעזרת המפתח הציבורי של השרת אלא בדיקה אל מול גורם שלישי (לרב חיצוני אך הדבר לא מחייב!) כי אותה חתימה אכן שייכת לאותה הכתובת.]

אופן השימוש בדרך זו מתבצע לרוב אל מול גורם שלישי בתקשורת, לאחר שקיבלנו את ה-Certificate מהשרת, אנו מבצעים בדיקה כי אכן ה-Fingerprint שלה זהה ל-Fingerprint שלה אצל הגורם השלישי (אצל הספק של אותה התעודה) ואם כן- אנחנו יכולים להיות בטוחים כי אכן מדובר ב-Certificate אמיתי ואין שום בעיה להשתמש במפתח הציבורי שהיא כוללת ולהצפין בעזרתו את המידע הרגיש שלנו שאותו נשלח בעתיד לשרת.

ה-Fingerprint של ה-Certificate הוא מעין ה-Hash של התעודה, והוא ייחודי לכל תעודה ותעודה, במקרה ותוקף או כל גורם אחר ינסה לבצע שינוי, ולו הקל ביותר, בשלב השני (שלב החלפת המפתחות) ה-Fingerprint של ה-Certificate ישתנה ולא יתאים לאותו Fingerprint שקיים אצל ספק התעודות (הגורם השלישי), מה שיגרום לכשל במהלך ה-Certificate Authority.

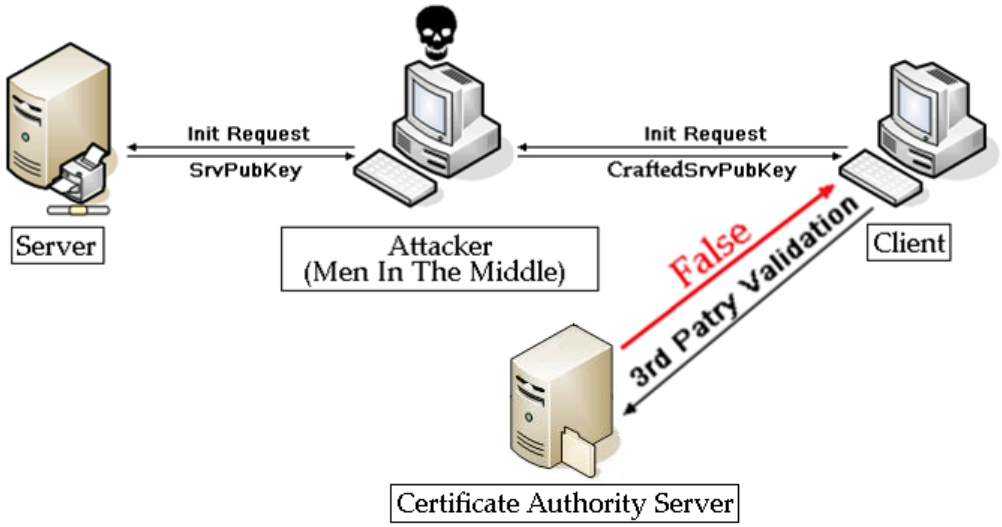
- **שלב שלישי -** הלקוח מתשאל צד שלישי (החברה המנפיקה של אותו Certificate) ומוודא כי הוא אכן אותנטי.



(שימו לב שבשלב הנוכחי הלקוח מתשאל את הצד השלישי בעזרת ה-Fingerprint של ה-CraftedSrvPubKey ולא של המפתח המקורי של השרת!)

הצד השלישי ישווה את ה-Fingerprint שהוא קיבל מהלקוח מול מסד הנתונים של שאר החתימות שלו ויבין שמישהו ערך את התעודה ושינה בה נתונים, במקרה כזה, השרת יחזיר תשובה שלילית ללקוח.

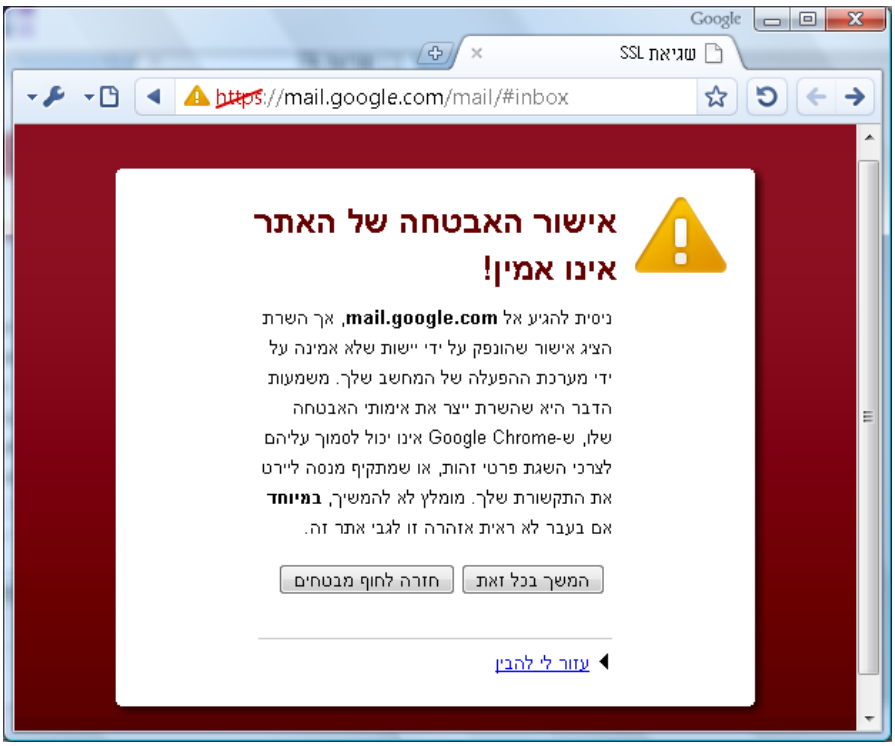
- **שלב רביעי -** הלקוח מקבל תשובה שלילית מהגורם השלישי, עקב כך שהחתימה של התעודה אינה תקינה.



במקרה וקיבלנו תשובה שלילית משרת ה-CA שלנו התקשורת נסגרת. שימו לב שעד כאן לא עבר שום מידע חסוי, המפתח הציבורי של השרת אליו רצינו לשלוח את המידע החסוי הוא לא מידע חסוי, התעודה והשרת שמולו ביצענו את הבדיקה גם הם לא מידע חסוי, הצלחנו לעצור את התקשורת לפני שהתוקף הצליח להשיג מידע חסוי.

נוכל לראות דוגמא נחמדה למקרה כזה אם למשל נגלוש באתר אינטרנט התומך במנגנון SSL כשאנחנו משתמשים בתוכנה Burp – שהיא מעין פרוקסי בין תוכנת הדפדפן שלנו לבין האינטרנט, התוכנה מאפשרת לנו לבצע מניפולציות על המידע העובר בין תוכנת הדפדפן לבין האינטרנט, ובמקרה שמדובר בתקשורת מול אתר התומך ב-SSL, התוכנה מפענחת את המידע, מציגה אותו למשתמש, וכאשר הוא מבקש לבצע את שליחת המידע- היא חותמת עליו בעזרת מפתח שלה (אין לה יכולת לחתום עליו בעזרת התעודה של השרת מפני שאין לה את המפתח הפרטי של השרת) ושולחת אותו לדפדפן- לאחר שהמידע הגיע לדפדפן, הדפדפן מזהה כי המידע נערך ומודיע על כך למשתמש.

דוגמא לניסיון התחברות לשרת הדוא"ל של גוגל, הפועל תחת פרוטוקול SSL בעזרת חיבור הנתון תחת ההזנה:



כיום כמעט (או אולי אפילו כולם?) כל הדפדפנים תומכים בהודעה מסוג זה, ההודעה באה להודיע למשתמש כי מישהו, היכנשהו, במהלך התקשורת שינה/חתם מחדש על התעודה שהונפקה ע"י גוגל.

סיכום + קישורים

אני מקווה שהבנתם איך הולך כל הרעיון של ה-SSL, למה הוא נועד, ואיך הוא פותר לנו את בעיית ההזנה/ כיום קיימות מספר מתקפות-Men In The Middle אשר מסוגלות לעקוף מנגנון זה, אך הדבר לא פשוט כל כך והוא דורש תחכום רב.

קישורים רלוונטיים:

- http://www.openssl.org - האתר של OpenSSL – כיום הספריה הגדולה והמתקדמת ביותר (בקוד פתוח תחת GPL) למימוש הפרוטוקול.
- http://www.ietf.org/rfc/rfc2818.txt - ה-RFC של השימוש ב-"HTTP Over TLS".
- http://www.webstart.com/jed/papers/HRM/references/ssl.html - מאמר ישן אבל טוב על הפרוטוקול SSL.
- http://www.kegel.com/ssl/api.html - מאמר על ה-APIs של ה-OpenSSL.