

הטכניון - מכון טכנולוגי לישראל  
הפקולטה להנדסת חשמל  
המעבדה למערכות תוכנה מרושתות

דו"ח סיכום פרוייקט

**שיפור אבטחת מידע באינטרנט מעל TLS**

הוד רז  
שון סעדון

מנחה:  
שי ורגפטיק

תאריך הגשה: נובמבר 2018

תוכן עניינים

[פרק 1 – מבוא 3](#_Toc529671108)

[1.1 הקדמה 3](#_Toc529671109)

[1.2 מטרת הפרוייקט 4](#_Toc529671110)

[פרק 2 - רקע תיאורטי 5](#_Toc529671111)

[2.1 מהי הצפנה א-סימטרית? 5](#_Toc529671114)

[2.1.1 דוגמה לאלגוריתם הצפנה א-סימטרי *Diffie-Hellman* 6](#_Toc529671115)

[2.2 מהי הצפנה סימטרית? 8](#_Toc529671118)

[2.2.1 דוגמה: *AES*, *Block cipher* & *CBC* 8](#_Toc529671119)

[2.3 מהו פרוטוקול ה-*TLS*? 10](#_Toc529671120)

[2.3.1 מוטיבציה לשימוש ב-*TLS* 10](#_Toc529671121)

[2.3.2 היסטוריה על קצה המזלג 10](#_Toc529671122)

[2.4 חבילת צופן (*Cipher Suite*) 11](#_Toc529671123)

[2.5 לחיצת היד של פרוטוקול ה-*TLS* 13](#_Toc529671124)

[2.5.1 יצירת המפתחות (*Key Generation*) 14](#_Toc529671125)

[פרק 3 – הבעיה הקיימת 17](#_Toc529671126)

[3.1 הקדמה 17](#_Toc529671127)

[3.2 דיון 18](#_Toc529671128)

[פרק 4 – הפתרון המוצע 19](#_Toc529671129)

[4.1 הרעיון הכללי 19](#_Toc529671130)

[4.2 שיפור האבטחה 20](#_Toc529671131)

[4.3 יישום הפתרון 20](#_Toc529671132)

[פרק 5 – בניית המערכת 22](#_Toc529671133)

[5.1 התשתית 22](#_Toc529671134)

[5.2 טופולוגיית המערכת 24](#_Toc529671135)

[5.3 תצורת המכונות הוירטואליות 24](#_Toc529671136)

[5.3.1 תצורת הלקוח 25](#_Toc529671137)

[5.3.2 תצורת ה-*middlebox* 25](#_Toc529671138)

[5.3.3 תצורת השרת 25](#_Toc529671139)

[פרק 6 – השינויים בקוד 26](#_Toc529671140)

[6.1 התאמת קוד ה-*TLS* 26](#_Toc529671141)

[6.2 התאמת קוד ה-*Proxy* 29](#_Toc529671142)

[פרק 7 – סיכום ורעיונות להמשך 33](#_Toc529671143)

[טבלת איורים 34](#_Toc529671144)

[ביבליוגרפיה 35](#_Toc529671145)

[*נספחים* 36](#_Toc529671146)

[נספח א': בדיקת תקשורת בין המכונות הוירטואליות 36](#_Toc529671147)

[נספח ב': בדיקת תקשורת בין המכונות הוירטואליות דרך שרת הפרוקסי 38](#_Toc529671148)

[נספח ג': קיבוע חבילת ההצפנה על גבי שרת ה-*Apache* 39](#_Toc529671149)

[נספח ד': פקודות לקימפול קוד ה-*TLS* 39](#_Toc529671150)

[נספח ה': הוספת ספריית הקריפטוגרפיה של *Python* 39](#_Toc529671151)

# פרק 1 – מבוא

## 1.1 הקדמה

התקשורת באינטרנט, כפי שנוצרה לראשונה, אינה עוד. כבר לא מדובר בשתי נקודות קצה המחליפות ביניהן מידע על גבי תווך פשוט המסוגל להעביר חבילות מידע ותו לא. המידע הנשלח מעובד לעיתים תכופות על ידי נקודות מתווכות (ייקראו מעתה *middleboxes*) כמו מערכות לגילוי פריצה, סורקי וירוסים, או חומות אש.

כשם שהשימוש בהצפנה ברשת עולה עם הזמן (נכון לשנת 2014, כמעט מחצית מרשת האינטרנט משתמשת בפרוטוקול *HTTPS*), כך אותן *middleboxes* גדלות במספרן, הופכות "עיוורות" ואינן יכולות לבצע את עבודתן.

על כן, אנו נדרשים לשאלה, כיצד משלבים את אותן ישוייות בתקשורת שנועדה, במקורה, להתבצע בין שתי נקודות קצה בלבד מבלי להתפשר ברמת האבטחה והאמינות.

## 1.2 מטרת הפרוייקט

יצירת תקשורת המתבססת על פרוטוקול האבטחה הנפוץ ביותר כיום ‒ *TLS*, כזו שתהיה מאובטחת בצורה מלאה מקצה לקצה (מודל לקוח ‒ שרת) תוך שמירה על שלושת העקרונות המנחים הבאים:

* פרטיות המידע - תוקף פוטנציאלי לא יוכל לקרוא את המידע העובר בין שתי נקודות הקצה
* אותנטיות המידע - תוקף פוטנציאלי לא יוכל לשנות, למחוק או להוסיף מידע
* אימות היישות - כל יישות אשר פונה אל משתמש הקצה מחוייבת להזדהות בפניו על מנת לקבל את מפתחות השיחה (*session keys*)

במילים אחרות, מטרתנו לאפשר תקשורת מאובטחת על גבי פרוטוקול *TLS* המתאימה לצרכים המודרניים הכוללים שימוש ב-*middleboxes* מבלי להפר את עיקרון קצה לקצה.

# פרק 2 - רקע תיאורטי

בפרק זה נרחיב על מושגים, פרוטוקולים ואלגוריתמים שעשויים להיות נחוצים להבנה מלאה של ספר הפרוייקט.



## מהי הצפנה א-סימטרית?

הצפנה א-סימטרית (Asymmetric encryption) היא הצפנה בה המפתח המשמש להצפנה שונה מזה המשמש לפענוח, זאת להבדיל מהצפנה סימטרית בה המפתח המשמש להצפנה הוא גם זה המשמש לפענוח.  
עקרון הפעולה הוא שבהינתן שני צדדים (להלן: **לקוח ושרת**) המעוניינים לתקשר בינהם בצורה מוצפנת, גם הלקוח וגם השרת יוצרים לעצמם בנפרד שני מפתחות הצפנה, פרטי וציבורי. המפתח הציבורי הוא מפתח הנגיש לכל ומשמש להצפנה. המפתח הפרטי הוא מפתח הנשמר בסוד והוא משמש לפענוח ההצפנה שנעשתה ע"י המפתח הציבורי. על מנת שהלקוח יוכל להעביר לשרת מידע מוצפן, הוא מקבל מהשרת את המפתח הציבורי שלו, מצפין איתו את המידע ושולח את המידע המוצפן בחזרה לשרת. השרת מקבל את המידע המוצפן וכאמור יפענח את המידע בעזרת המפתח הפרטי המתאים שברשותו.

הצפנה א-סימטרית איטית באופן משמעותי בהשוואה להצפנה סימטרית ולכן המוטיבציה לשימוש בהצפנה סימטרית גבוהה יותר.

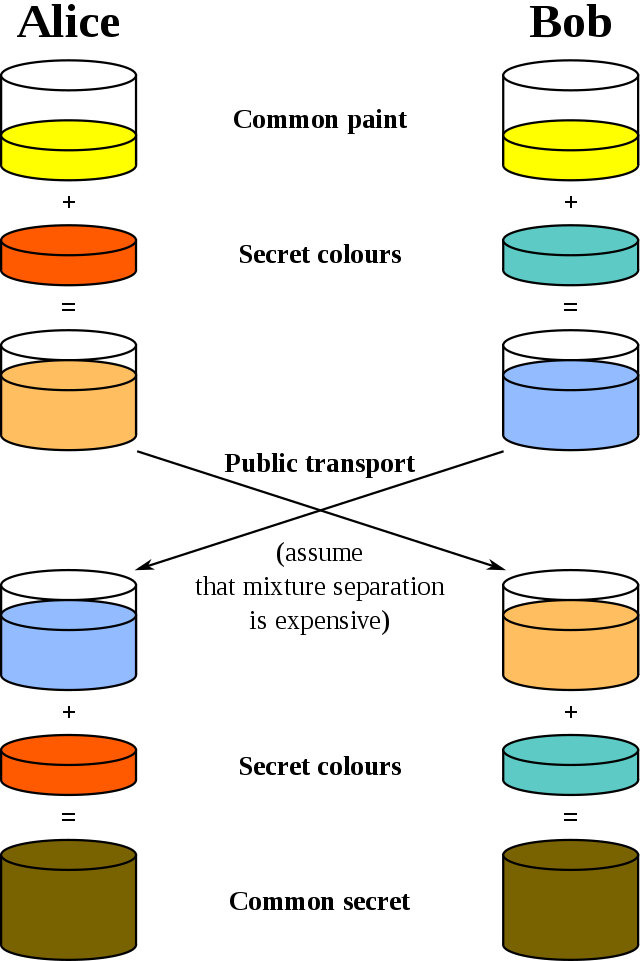
הבעיה העולה היא, כיצד ניתן להעביר באופן בטוח את אותו המפתח כדי שבשני צדדים מרוחקים יהיו את אותם המפתחות הסימטריים? כאן בא לעזרתנו אלגוריתם *Diffie-Hellman*.[[1]](#footnote-1)

### דוגמה לאלגוריתם הצפנה א-סימטרי *Diffie-Hellman*

ראשית, למען ביסוס הבנת האלגוריתם, להלן הסבר אודות מהו מפתח קריפטוגרפי.  
מפתח קריפטוגרפי (*Cryptographic Key*) הוא מחרוזת של ביטים הנוצרת עקב שימוש באלגוריתם קריפטוגרפי לשימושים שונים. אחת הדוגמאות החשובות (והטריוויאליות) למפתח זה הוא מפתח הצפנה, שתפקידו להפוך מידע גלוי למוצפן ו/או ההיפך. קיימים סוגים שונים של מפתחות קריפטוגרפיים עליהם נרחיב בהמשך.

אלגוריתם *Diffie*-*Hellman*, או בקיצור, *DH* הוא פרוטוקול להחלפת מפתחות קריפטוגרפיים בין שתי נקודות קצה מעל תווך גלוי (שאינו מוצפן). המתמטיקה מאחורי האלגוריתם אינה מסובכת להבנה, אך לא ניכנס כאן לפרטים מתמטיים ונתמקד בהבנה אינטואיטיבית.  
נאמר שהוד ושון רוצים לתקשר אחד עם השני בלי שהמנחה שלהם, שי, יידע מה הם אומרים, על אף שהוא מאזין לכל ההתקשרות ביניהם. כלומר, כל מה שהוד ישלח לשון, יתקבל אצל שי ובאופן דומה שי יקבל את כל מה ששון ישלח להוד. אז כיצד הוד ושון יוכלו לתקשר מבלי ששי יבין?  
כאן *Diffie-Hellman* נכנס לתמונה.  
ניתן להבין את פעולת האלגוריתם בצורה הכי פשוטה ואינטואיטיבית באמצעות ערבוב צבעים.

להלן איור המתאר את פעולת אלגוריתם *Diffie-Hellman* בין אליס ובוב.



##### *איור 1: תיאור פעולת אלגוריתם Diffie-Hellman*

כעת, נסביר את האיור, בו אליס ובוב רוצים לתקשר זה עם זה ולהגיע לסוד משותף:

* התהליך מתחיל בכך ששני הצדדים, אליס ובוב, מסכימים על צבע ראשוני שרירותי שאינו צריך להישמר בסוד (אבל צריך להיות שונה בכל פעם). באיור זהו הצבע הצהוב.
* בשלב הבא, כל אחד מהם בוחר צבע סודי (מקביל למפתח הפרטי) שהוא שומר לעצמו - במקרה זה, כתום (עבור אליס) וכחול (עבור בוב). החלק המכריע של התהליך הוא שגם אליס וגם בוב מערבבים את הצבע הסודי שלהם יחד עם הצבע המשותף, וכתוצאה מכך יוצרים תערובת מיוחדת, אותה מחליפים באופן ציבורי.

**הערה**: יש לציין כי האלגוריתם מסתמך, כאשר אנו עושים הקבלה לצבעים, על העלות הגבוהה שבהפרדת הצבעים למרכיביהם המקוריים, או אם נרצה, מספר האפשרויות העצום שצריך לנסות על מנת להגיע בדיוק לצבעים המקוריים, כך שבהסתברות גבוהה, לא ניתן לעשות זאת בזמן סביר.

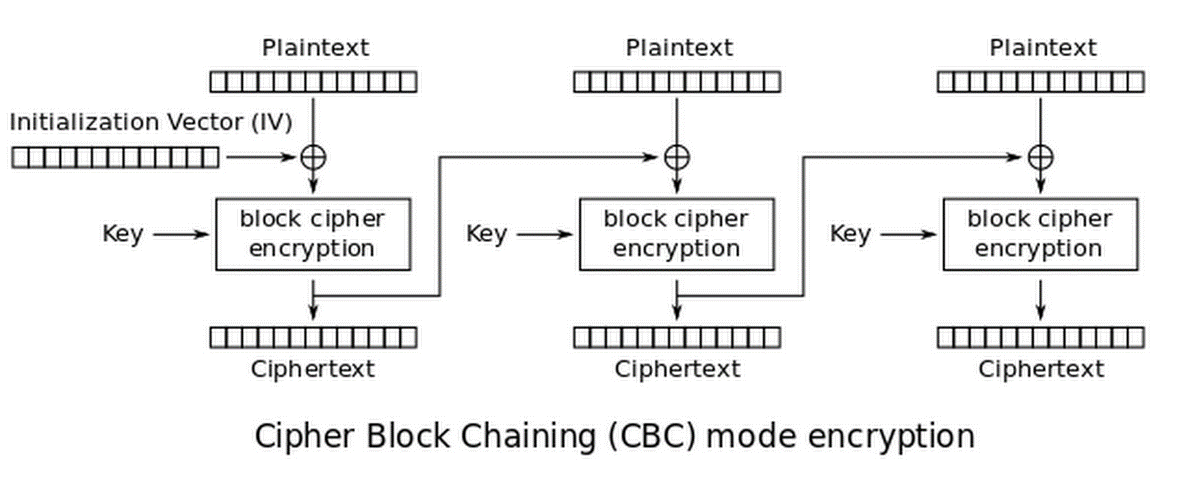
* לבסוף, כל אחד מהם מערבב את הצבע שהתקבל מהאחר עם הצבע הפרטי והסודי שלו.   
  התוצאה היא תערובת צבע סופי (ירוק זית כהה), תוצאה זהה לתערובת הצבע הסופי של השותף.   
  בהתבסס על כך שבערבוב צבעים אין חשיבות לסדר ערבוב הצבעים, או במונחים מתמטיים, זוהי פעולה קומוטטיבית.[[2]](#footnote-2)



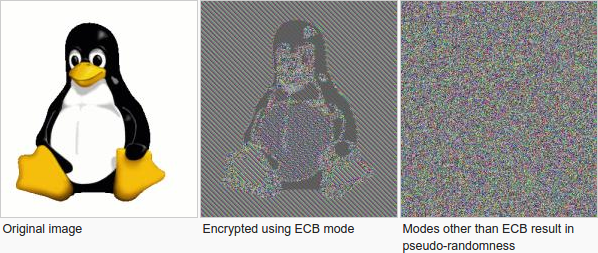
## מהי הצפנה סימטרית?

הצפנה סימטרית, כאמור, היא הצפנה בה המפתח המצפין הוא גם המפענח. על כן, בהינתן שני צדדים המעוניינים לתקשר בינהם בצורה מוצפנת, שניהם צריכים להחזיק במפתח זהה שישמש אותם להצפנה בעת שליחת מידע ולפענוח בעת קבלתו.

### דוגמה: *AES*, *Block cipher* & *CBC*

* צופן בלוקים (*Block* *cipher*) – אלגוריתם הצפנה שמצפין מספר קבוע של ביטים של מידע הנקרא בלוק. גודל הבלוק עשוי להכיל 64, 128 או 256 ביטים.   
  לדוגמה, יהי בלוק בגודל 128 ביטים של טקסט גלוי (לא מוצפן), לאחר הצפנה יהפוך לבלוק בגודל 128 ביטים של טקסט מוצפן.   
  במידה והמידע שנדרש להצפין קטן מ - 64 ביטים, יתבצע ריפוד (הוספת ביטים להשלמת 64 ביטים).  
  הרוב הגדול של הצפנים הסימטריים הם מסוג צופן בלוקים ואחד הנפוצים שבהם הוא *AES*.[[3]](#footnote-3)
* *Advances* *Encryption* *Standard (AES)* – זהו תקן הצפנה סימטרי שהחליף בשנת 2002 את התקן *DES* שפותח בסוף שנות ה-70. ל- *AES* קיימים אלגוריתמים המגדירים כיצד להפעיל אותו, וכקבוצה נקראים אופני ההפעלה (*Mode* *of operation*) של צופן בלוקים. אחד מאופני ההפעלה הוא אופן *CBC*.[[4]](#footnote-4)
* וקטור אתחול (*Initialization Vector*) - מספר אקראי ייחודי המשמש לאתחול את תהליך ההצפנה באלגוריתמים מסוימים.
* אופן הפעולה *Cipher-Block Chaining (CBC)* - מחלק את הטקסט הגלוי לבלוקים בגודל הרצוי, יוצר וקטור אתחול (*IV*) בגודל הרצוי ומפעיל פעולת *XOR* בין ה-*IV* לבין הבלוק הראשון. על התוצאה המתקבלת מופעל *XOR* עם הבלוק הבא וכן הלאה עד סיום בלוקי המידע.  
  ישנם אופני פעולה שונים ואחד מהם הוא *ECB*. אופן זה עושה פעולה זהה לאופן *CBC*, אך אינו משתמש בוקטור אתחול. תכונה זו עומדת לו לרועץ, כך שבהינתן ידע כלשהו על המידע המועבר, ניתן לפענח את המידע המוצפן ועל כן כמעט ולא משתמשים באופן זה בימינו.  
    
  להלן דיאגרמה הממחישה את הליך ההצפנה באופן הפעולה *CBC*:

##### *איור 2: תיאור פעולת אלגוריתם CBC*

להלן איור המראה חולשה נוספת ב-*ECB* לעומת *CBC* המשתמש בוקטור אתחול:

##### *איור 3: דוגמה לחולשה באלגוריתם EBC לעומת CBC*

## מהו פרוטוקול ה-*TLS*?

אבטחת שכבת התעבורה (*Transport* *Layer* *Secure*) הוא פרוטוקול אבטחה באינטרנט, שנועד לשמור על שלושת העקרונות הבאים: פרטיות, אמינות ושלמות נתונים בין שני יישומי תקשורת.   
זהו פרוטוקול האבטחה הנפוץ ביותר שבו נעשה שימוש כיום והוא בעיקר משמש דפדפני אינטרנט ויישומים אחרים הדורשים העברת נתונים בצורה מאובטחת ברשת, כגון העברות קבצים, חיבורי *VPN*, הודעות מיידיות ועוד.[[5]](#footnote-5)

### מוטיבציה לשימוש ב-*TLS*

נרצה להעביר מידע בין שני מחשבים בצורה בטוחה, כלומר למנוע מקרים כמו גניבה, זיוף או חבלה במידע שאנו מעבירים בצורה יעילה.

### היסטוריה על קצה המזלג

ראשיתו של הפרוטוקול הגיע בכלל מפרוטוקל אחר, שהוא אב קדמון ל-*TLS* והוא נקרא  
*Secure* *Sockets* Layer או בקיצור *SSL*. פרוטוקול ה-*SSL* הומצא ע"י חברת *NetScape* וזאת על מנת לספק רמת אבטחה לדפדפן שהיא יצרה, *NetScape* *Navigator*, שאינו קיים עוד היום. ל-*SSL* מספר גרסאות, כאשר הראשונה נגנזה עוד בשלב הפיתוח של הפרוטוקול וגרסותיה האחרות שוחררו במחצית השניה של שנות ה-90.   
בשנת 1999 שוחררה הגרסה הראשונה של פרוטוקול ה- *Transport* *Layer* *Protocol* או בקיצור *TLS*, שהוא אינו אלא שיפור קל של פרוטוקול ה-*SSL* מגרסה 3 שקדמה לו.

מאז ועד היום אנו משתמשים בגרסאות *TLS* כאלו ואחרות על מנת לאבטח את המידע שאנחנו שולחים ומקבלים מהמחשב שלנו. השינוי בין הגרסאות נובע בעיקר מהוספת שיפורים או לחילופין, ליקויי אבטחה שנמצאו בפרוטוקול.

נכון לכתיבת שורות אלו, רוב תקשורת ה-*TLS* בעולם נעשית בעזרת *TLS v1.2* ובימים אלו נעשים מאמצים להטמעת הגרסה *TLS v1.3*.

## חבילת צופן (*Cipher Suite*)

זוהי ערכה של אלגוריתמים קריפטוגרפיים, הנועדה להגן על המידע העובר בחיבור בין שני מחשבים, למשל בתקשורת מעל פרוטוקול *TLS*.   
בדרך כלל חבילת הצופן תכיל אלגוריתמים לשלוש הבעיות (המתאימות לשלושת העקרונות המנחים של הפרוטוקול שהוזכרו לעיל):

* החלפת מפתחות (*Key Exchang*e)
* הצפנת המידע (*Bulk* *Encryption*)
* אמינות המידע (*Message* *Authentication* *Code*)

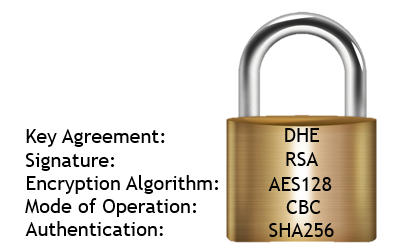
מספר האפשרויות לבחירת חבילת הצופן הוא רב והיא נעשית בזמן לחיצת היד של פרוטוקול ה- *TLS*.

דוגמה של חבילת צופן שנבחר למימוש :



##### *איור 4: דוגמה לחבילת צופן*

להלן איור המראה את השדות המרכיבים את חבילת הצופן ומשמעותם:

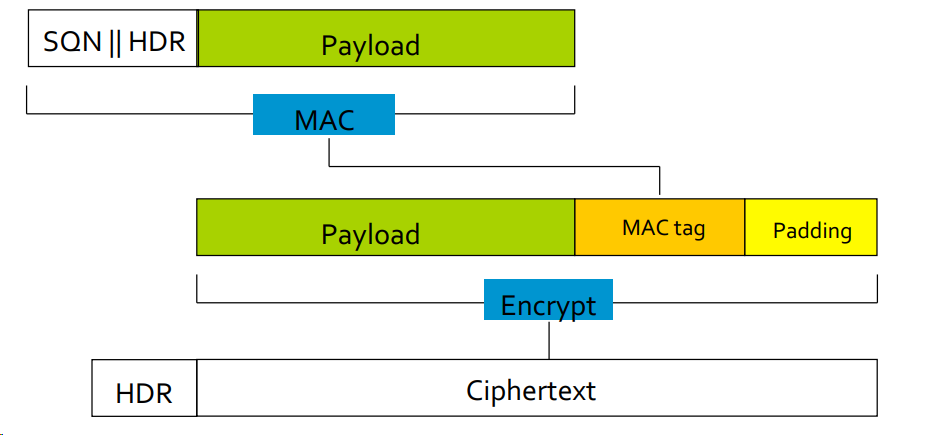


##### *איור 5: השדות בחבילת הצופן*

חבילת הצופן היא זו שמתארת את האופן שבו יווצרו המפתחות הקריפטוגרפיים. מספר המפתחות הקריפטוגרפיים בסיום התהליך הוא 4, אך כתלות בבחירת חבילת צופן, עשויים להיווצר שני מפתחות נוספים והם שני וקטורי אתחול.

מפתח ה-*MAC* - קוד אימות מסרים (*Message Authentication Code*), הוא שם כולל לאוסף של פונקציית גיבוב (*hash*) קריפטוגרפיות חד-כיווניות. תכליתו לשמור על אחד משלוש העקרונות של ה-*TLS*, כפי שראינו בתחילת הפרק והוא עקרון האימות.   
כאשר מתקבלת הודעה אצל מחשב מסוים, צריכה להיות לו האפשרות לדעת האם ההודעה שהגיעה אליו היא אכן ההודעה המקורית כפי שהמחשב השולח התכוון לשלוח. הדרך לוודא זאת היא באמצעות מפתח זה ודרך פעולתו היא כדלקמן:  
כל הודעה הנשלחת מצד אחד לשני נחתמת על ידי השולח, כך שהצד המקבל יוכל להפעיל את אותה הפונקציה על ההודעה שהתקבלה ולבדוק האם ההודעה שהתקבלה היא אותנטית. אם היא שונתה ע"י גורם באמצע, ההסתברות לכך שהצד המקבל יבחין בכך היא כמעט ודאית ובמקרה כזה היא תדאג לעדכן בכך את השולח.   
לקבלת אינטואיציה חזקה יותר, ניתן לחשוב על *Checksum* כבדיקה בעלת רעיון דומה, שנעשית במקרים מסוימים.5

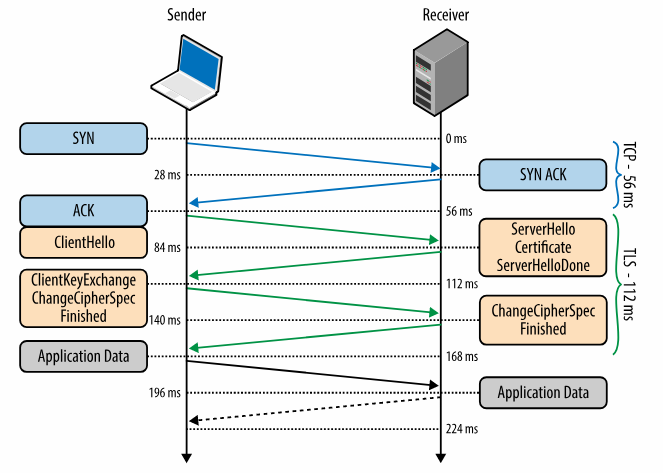
אלגוריתם *HMAC* הוא דוגמה לפונקציית גיבוב חד-כיוונית שבה ראשית כל הטקסט הגלוי עובר בפונקציית הגיבוב ורק לאחר מכן מוצפן ע"י מפתח הצפנה.



##### *איור 6: פעולת החתימה ולאחריה ההצפנה*

## לחיצת היד של פרוטוקול ה-*TLS*

להלן דיאגרמה המתארת את לחיצת היד (*Handshake*):[[6]](#footnote-6)



##### *איור 7: סכימה לתיאור לחיצת היד של פרוטוקול ה-TLS*

הסבר אודות הדיאגרמה:

* עם שליחת ה-*Ack* האחרון, המורה על סיום תהליך ה-*Handshake* של פרוטוקול ה-*TCP* (בכחול), הלקוח מתחיל את תהליך ה-*Handshake* של פרוטוקול ה-*TLS* (בכתום).
* הלקוח שולח לשרת הודעת *ClientHello* ובה את המידע על גרסת ה-*TLS* בה יכול הלקוח לתמוך, רשימת *Cipher* *Suites* בהן הוא תומך וכמו כן נשלחים גם פרמטרים נוספים.
* בתגובה, משיב השרת בהודעת *ServerHello* ובה מורה ללקוח על החלטתו בעניין בחירת הגרסה של   
  ה-*TLS*, באיזו חבילת הצפנה (*Cipher* Suite) עליו להשתמש, תעודת מפתח ציבורי (*Certificate*) וכן פרמטרים נוספים.  
  לאחר שהלקוח מקבל הודעה זו מהשרת, הוא מאמת את התעודה שקיבל ויכול ליצור אצלו את המפתחות הקריפטוגרפיים שישמשו לאחר מכן גם את הלקוח וגם את השרת.
* בשלב הבא הלקוח מחזיר הודעה ובה הוא מעביר לשרת מפתח אקראי חדש הנקרא *pre-master secret* שנוצר אצלו כאשר הוא מוצפן בעזרת המפתח הציבורי של השרת.
* בשלב ה- *Change Cipher Spec* שני הצדדים גוזרים מפתחות קריפטוגרפיים סימטריים, מתוך   
  ה- *pre-master secret* ופרמטרים נוספים.  
  בנוסף, הלקוח משתמש במפתחות הצפנה אלו על מנת להצפין את הטקסט *“Client Finished”* וזאת על מנת שהשרת שמקבל את ההודעה יוכל להשתמש במפתחות לפענוח ולוודא שהוא אכן מקבל את המחרוזת הנ"ל. זוהי מעין "בדיקת שפיות" מובנית בתוך הקמת התקשורת המאובטחת.
* בשלב הסופי (הכתום האחרון) השרת יבדוק את הפרמטרים שנשלחו מהלקוח ובמידה והכל תקין הוא ישלח ללקוח הודעה מוצפנת *“Server Finished”*.
* מעתה ואילך (באפור) תוכן הודעת ה- *Application Data* תהיה כבר מידע מוצפן שהלקוח מעוניין להעביר לשרת.[[7]](#footnote-7)

### יצירת המפתחות (*Key Generation*)

בעת תהליך לחיצת היד, מיד לאחר שליחת הודעת ה- *ServerHello* וכתלות בחבילת הצופן הנבחרת, הלקוח מקבל לידיו בלוק שנקרא *Pre-Master Secret*.

#### Pre-Master Secret

לאחר תהליך החלפת המפתחות, יהיה גם בצד השרת וגם בצד הלקוח אותו המידע הנקרא*Pre-Master Secret*. כעת, מתבצע חישוב מתמטי המשלב פרמטרים שהתקבלו בתהליך לחיצת היד לתוך פונקציה פסאודו-אקראית (*Pseudo-Random Function*) או בקיצור *PRF*, והם יוצרים בלוק חדש הנקרא *Master* *Secret*.

#### Master Secret

זהו בלוק בן 48 בתים והוא מכיל 4 מפתחות:

* *client\_write\_MAC\_key* המשמש גם כ *server\_read\_MAC\_key*
* *server\_write\_MAC\_key* המשמש גם כ *client\_read\_MAC\_key*
* *client\_write\_key* המשמש גם כ *server\_read\_key*
* *server\_write\_key*  המשמש גם כ *client\_read\_key*

**הערה**: לשם ההבהרה, כל נקודה מציינת מפתח יחיד ולכל מפתח ישנם שני שמות וזאת מפני שכל מפתח הוא סימטרי. לדוגמה, המפתח השלישי ישמש את הלקוח להצפנת מידע ואותו מפתח בדיוק ישמש את השרת לפענח את המידע המוצפן שהלקוח שלח לו.

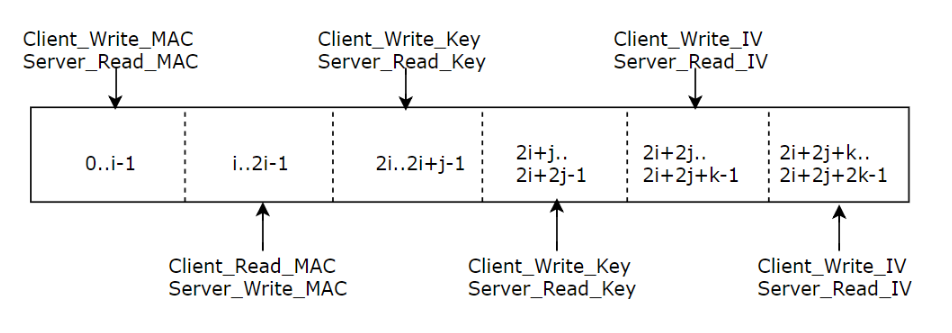
כעת, בדומה לסעיף הקודם, הלקוח והשרת מבצעים חישוב מתמטי נוסף ובאמצעות שימוש בפונקציית *PRF* בשנית, נוצר בלוק המפתחות.

#### בלוק מפתחות (Key Block)

זהו הבלוק הסופי ממנו נגזרים המפתחות הקריפטוגרפיים. בחבילות צופן מסוימות המשתמשות למשל באופן *CBC*, מתווספים שני מפתחות הצפנה נוספים, והם וקטורי האתחול.   
לכן, בסך הכל נקבל בלוק שממנו נבצע את גזירת 6 המפתחות והם:[[8]](#footnote-8)

* *client\_write\_MAC\_key* המשמש גם כ *server\_read\_MAC\_key*
* *server\_write\_MAC\_key* המשמש גם כ *client\_read\_MAC\_key*
* *client\_write\_key*  המשמש גם כ *server\_read\_key*
* *server\_write\_key* המשמש גם כ *client\_read\_key*
* *client\_write\_IV\_key* המשמש גם כ *server\_read\_IV\_key*
* *server\_write\_IV\_key* המשמש גם כ *client\_read\_IV\_key*

להלן איור המתאר את חלוקת הבלוק למפתחות בעזרת האינדקסים *i, j, k* כאשר *i* הוא גודל מפתח ה-*MAC*, *j* הוא גודל מפתח ההצפנה ו-*k* הוא גודל וקטור האתחול.



##### *איור 8: חלוקת הבלוק למפתחות*

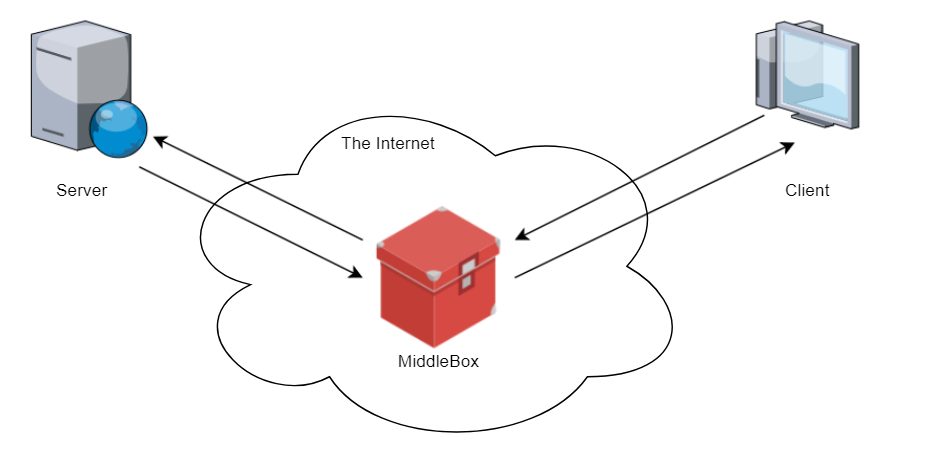
# פרק 3 – הבעיה הקיימת

## 3.1 הקדמה

כאמור, תקשורת אינטרנט מאובטחת המתבססת על פרוטוקול *TLS*, תוכננה עבור שני צדדים בדיוק.   
לכן, כיום, כאשר אנו מנהלים תקשורת מאובטחת על גבי *TLS* מול שרתים של חברות כדוגמת גוגל, אמאזון, או פייסבוק, נהוג לבצע הליך בשם *"TLS Split"*.

הליך זה מפצל את התקשורת המאובטחת לשני חיבורים מאובטחים מקביליים. האחד, בו ה-*middlebox* "מתחזה" לשרת בפני הלקוח (בעזרת מפתח ההצפנה הפרטי של השרת), והשני, בו ה-*middlebox* מחובר לשרת.   
בכך למעשה, נחלשת האבטחה באופן דרסטי, בין השאר משום שהלקוח אינו יכול לאמת את ה-*middlebox* באופן מפורש או להיות בטוח שה-*middlebox* יאמת כראוי את השרת.

האיור הבא מציג (בצורה מופשטת) את הבעיה. נבחין כי זוג חיצים מייצג חיבור תקשורת מאובטחת על גבי *TLS*.



##### *איור 9: מבנה הרשת הקיימת*

## 3.2 דיון

הליך ה- *"TLS Split"* טומן בחובו בעיות אבטחה שכן תקשורת בין שני צדדים, לקוח ושרת, עוברת כעת דרך צד שלישי, צד אשר יכול להוות נקודת תורפה אבטחתית. נרצה לדון בבעיה זו.

שיחה מאובטחת בין שני צדדים יכולה להתקיים רק לאחר לחיצת היד (של פרוטוקול ה-*TLS*)ביניהם. לצורך הסכמה זו, נדרשות מפתחות הצפנה, תחילה מפתחות א-סימטריים ולבסוף מפתחות סימטריים.  
מטרת תהליך ההצפנה ה-א-סימטרי (כפי שפירטנו בהרחבה בפרק 2) הוא להגיע לסוד המשותף לשני הצדדים, ממנו יוכלו הצדדים המעורבים לחלץ את מפתחות ההצפנה הסימטריים וכך להתחיל לתקשר באופן מאובטח. לשם תהליך זה, כל אחד מהצדדים נדרש למפתח ציבורי, כזה שעומד לרשות הכלל (עבור צד הלקוח, דרישה זו לא נחוצה שכן הוא פונה אל השרת ולא להיפך), ומפתח פרטי, כזה שידוע רק לו, אחד שתואם את הציבורי וקשור אליו.  
אם כן, כיצד ייתכן שה- *middlebox* מסוגל להתחזות לשרת?

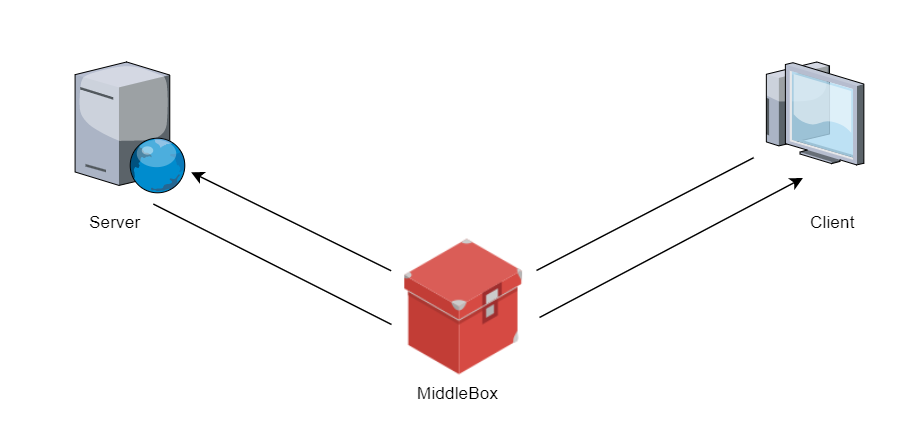
כאן טמונה הבעייתיות. ה- *middlebox* מסוגל להתחזות לשרת משום שהשרת מוסר לו את המפתח הפרטי. מסירת המפתח מפרה את עקרונות האבטחה הבסיסיים והיא כשלעצמה בעיה, אך זה לא נגמר כאן.   
כעת, תקשורת המתנהלת בין הלקוח לבין ה- *middlebox* (השרת מבחינתו של הלקוח) חשופה בפני צד שלישי, והמידע המועבר בין הלקוח ובין השרת נתון לחסדיו של ה- *middlebox*. אם ירצה לשנות, להוסיף או למחוק, יוכל לעשות זאת, שכן קיימים שני חיבורים מאובטחים שונים.

# פרק 4 – הפתרון המוצע

## 4.1 הרעיון הכללי

הקמת תקשורת מאובטחת מקצה לקצה, כזו המערבת שני צדדים בלבד, דורשת יצירת מפתחות קריפטוגרפיים סימטריים על מנת להצפין בצד אחד (ולפענח בצד שני) את המידע. לכן, על מנת שנוכל לשלב צד שלישי בתקשורת, ולעשות זאת בצורה מפוקחת מבלי להפר את אמון הלקוח או את עקרונות הקריפטוגרפיה הבסיסיים, נדרוש מה-*middlebox* להזדהות בפני הלקוח ולבקש את המפתחות הרלוונטיים (תלוי תכליתו של ה-*middlebox* ‒ קריאה בלבד עבור *firewall ,* קריאה וכתיבה עבור *compression proxy*).

כפי שתיארנו בפרק 2 בהרחבה, מפתחות ההצפנה (והפענוח) המופקים זהים בכל אחד מהצדדים, ולכן כשבידו המפתחות, יוכל ה-*middlebox* לבצע את תפקידו.   
  
עתה, קיים חיבור מאובטח בודד (בין הלקוח לשרת) כפי שנוכל לראות באיור הבא.   
נבחין כי זוג חיצים מייצג חיבור תקשורת מאובטחת על גבי *TLS*.



##### *איור 10: מבנה הרשת לפי הפתרון המוצע*

## 4.2 שיפור האבטחה

המפתח הפרטי של המכונה הוא חלק אינהרנטי בניהול תקשורת מאובטחת, ולמעשה אם יגיע לידיו של תוקף פוטנציאלי, יסתום את הגולל על יכולת זו. לכן, מסירת המפתח הפרטי של השרת ל-*middlebox* לטובת תפקידו פוגע ברמת האבטחה ועל כן ברצוננו להימנע מכך בכל מחיר.

הפתרון אותו אנו מציעים משתמש במפתחות השיחה (*session)* אותם הלקוח מוסר, מבחירה, לצד השלישי   
(ה-*middlebox*)ולא במפתח הפרטי של השרת, ובכך משפר את האבטחה ברשת. בנוסף, הלקוח נהנה מהידיעה ‒ כי קיים צד שלישי בשיחה זו, ולא משנית בחשיבותה, והבחירה ‒ האם ברצונו להעביר את המפתחות.  
**הערה**: הפתרון הוא סימטרי וניתן לחילופין להציע, שהשרת הוא זה שיחלוק את המפתחות עם ה-*middlebox.*

## 4.3 יישום הפתרון

ראשית, נרצה לציין כי הצורה בה המערכת נבנתה ועליה הפתרון מומש הינה בגדר *proof of concept* ,   
כלומר, ניתן היה לבצע זאת בדרך אחרת.

אם כן, נתאר את תהליך בניית המערכת. תחילה, הוקמו 3 מכונות וירטואליות בעזרת *VirtualBox*, אחת עבור כל אחד מהצדדים בתקשורת המאובטחת ‒ הלקוח, השרת וה-*middlebox*. בהמשך נוצרו 2 רשתות נפרדות, אחת המכילה את הלקוח ואת ה-*middlebox* והשניה המכילה את ה-*middlebox* ואת השרת, על מנת לדמות תרחיש מהעולם האמיתי בו המידע היוצא מהלקוח עובר מסלול ארוך (דרך *middlebox* אחד או יותר) עד אשר מגיע לבסוף לשרת.

בצד הלקוח, משוכתינו העיקרית היא חילוץ מפתחות השיחה הסימטריים שנוצרים בתהליך לחיצת היד בין הלקוח ובין השרת. לשם כך, קוד המקור של ה-*TLS* שונה כך שיתאים לצרכינו, ובעת יצירתם, כל ששת המפתחות הודפסו לקובץ על גבי המכונה. קוד המקור בו נעשה שימוש שייך לערכת הכלים של *OpenSSL* וגירסתו  
*openssl-1.0.2n*.

בצד ה-*middlebox* הוקם פרוקסי (*proxy*) ‒ מערכת תוכנה (במקרה שלנו) הרצה על גבי המכונה ופועלת כמתווך בין התקן נקודת קצה, כגון מחשב, לבין שרת אחר שממנו לקוח מבקש שירות. מטרת הפרוקסי היא לסייע בפענוח חבילות המידע העוברות בין הלקוח והשרת ולמשוך את המפתחות מהלקוח בעזרת העברת קבצים מאובטחת ‒  
*SSH secure file transfer* . הקוד המריץ את הפרוקסי כתוב בשפת *Python* בסביבת הפיתוח *PyCharm*.  
נרצה להדגיש כי במימוש הפתרון שלנו ה-*middlebox* לא מזדהה בפני הלקוח ולוקח את מפתחות השיחה מבלי לבקשם ראשית אך כפי שהוזכר לעיל, מדובר בהוכחת היתכנות.

בצד השרת, הוקם שרת אליו הלקוח יוכל לפנות בעזרת בקשת *HTTPS* ‒ בקשת *HTTP* על גבי פרוטוקול *TLS*; למטרה זו, הותקן *Apache Server* ‒ תוכנה המספקת שירותי *web server* בה בוצעו שינויים תצורתיים אשר מאפשרים את קיבוע חבילת ההצפנה (*Cipher Suite*) בה נעשה שימוש.   
פרטיה כלהלן:

*TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256  
Key Agreement: DHE  
Signature: RSA  
Encryption Algorithm: AES128  
Mode of Operation: CBC  
Authentication: SHA256*

כאמור, תקשורת מאובטחת המבוססת על פרוטוקול *TLS* מצריכה תעודה (*Certificate*) אותו מציג השרת בפני הלקוח כצורת הזדהות (ובה, כפי שמופיע בפרק 2 בהרחבה, מופיע המפתח הציבורי של השרת). לכן, נוצרה תעודה המכילה את פרטי השרת (כגון שם השרת, הארגון לו שייך, המדינה בה נמצא וכו') המאמתת את זהותו בפני הלקוח.

כעת, לאחר הקמת המערכת, הוקמה תקשורת בין הלקוח ובין השרת בעזרת הפקודה *openssl* , המורצת מספריית קוד המקור לאחר ששונה לצרכינו, עם הפרמטרים *s\_client -connect <server’s ip address>:443* .

אם נסכם, בעת הקמת התקשורת בין הלקוח לבין השרת, לא נבחין בהבדל בין התליכים עד אשר שלב יצירת המפתחות כחלק מתהליך לחיצת היד בין הצדדים. בשלב זה, ה-*middlebox* שולף את המפתחות מהלקוח אשר מאפשרים לו לקרוא את המידע העובר בין השרת והלקוח.

# פרק 5 – בניית המערכת

## 5.1 התשתית

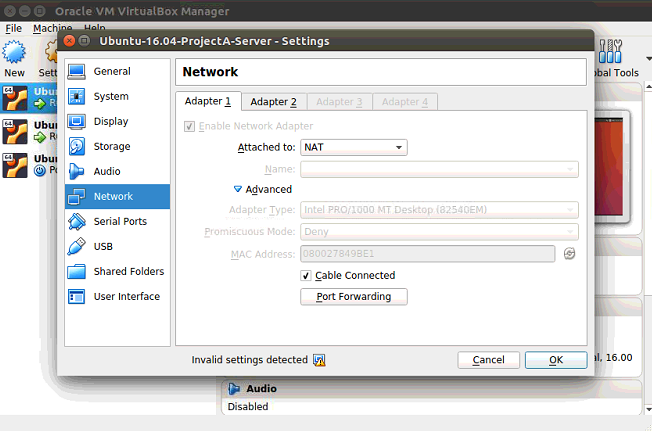
הפתרון שלנו דורש שלוש מכונות, אחת עבור צד הלקוח, השניה עבור צד השרת והשלישית עבור ה-*middlebox*. לצורך כך, הוקמו שלוש מכונות וירטואליות בעזרת התוכנה *VirtualBox* על גבי מחשב פיזי (כזה עם משאבים מתאימים).

נוסף על כך, נעשה שימוש ב-*VirtualBox Manager* על מנת לקבוע את תצורת המכונות הוירטואליות.

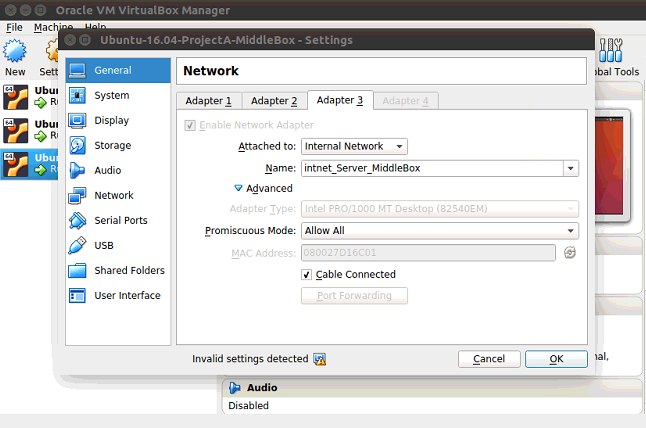
* המכונה המריצה את הלקוח נקראת *Ubuntu-16.04-ProjectA-Client*
* המכונה המריצה את ה-*middlebox* נקראת *Ubuntu-16.04-ProjectA-MiddleBox*
* המכונה המריצה את השרת נקראת *Ubuntu-16.04-ProjectA-Server*

הקמת התשתית כוללת בתוכה גם את תצורת מתאמי הרשת.

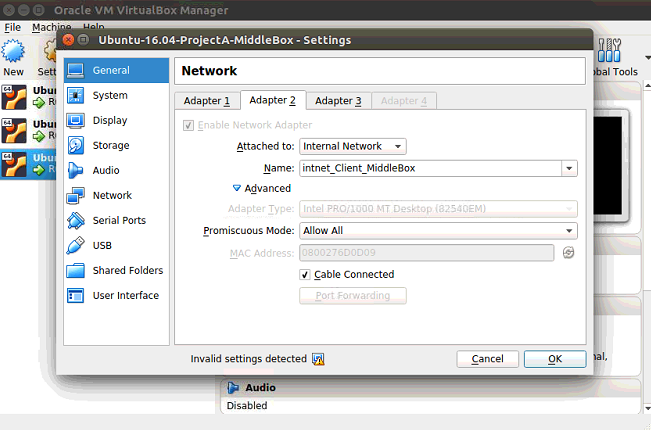
האיור הבא מציג את הגדרת ה-*NAT* עבור כל שלושת המכונות. נציין כי בכל השלושה, ה-*NAT* מוגדר באופן זהה.



##### *איור 11: הגדרת ה-NAT*

האיורים הבאים מציגים את הגדרת ה-*Internal* *Network* עבור כל שלושת המכונות. נציין כי במכונות הלקוח   
וה-*middlebox* שם הרשת הוא *intnet*\_*Client*\_*MiddleBox* ובמכונות השרת וה-*middlebox* שם הרשת   
הוא *intnet*\_*Server*\_*MiddleBox*.

##### *איור 12: הגדרת intnet\_Client\_MiddleBox*

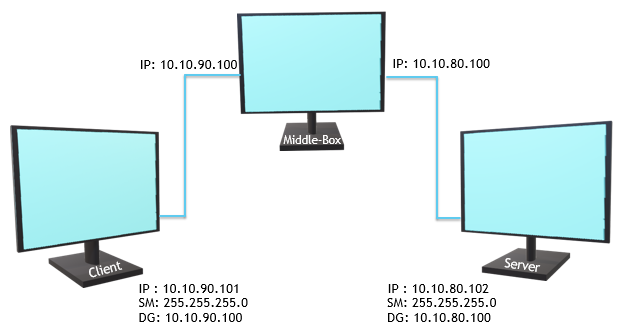


##### *איור 12: הגדרת intnet\_Client\_MiddleBox*

## 5.2 טופולוגיית המערכת

כאמור, המערכת מחולקת לשתי תת רשתות נפרדות, אחת מכילה את הלקוח ואת ה-*middlebox* והשניה מכילה את ה-*middlebox* והשרת. הסיבה למימוש זה היא שרצינו לפתור את הבעיה בצורה כללית, בה אנו ניגשים מתוך רשת אחת לאחרת ובכך לדמות סיטואציה "מהעולם האמיתי".

להלן איור המתאר את טופולוגיית הרשת שבנינו.



##### *איור 13: טופולוגיית הרשת הממומשת*

## 5.3 תצורת המכונות הוירטואליות

ניטור תעבורת המידע בין המכונות התבצע בעזרת *Wireshark*. נרצה לציין כי כלי זה תרם רבות להבנת התקשורת בין המכונות ובשלב מאוחר יותר בפרוייקט גם לניתוח חבילות המידע המתקבלות.

כעת, נסביר כיצד הוגדרו כל אחת מהמכונות. להרחבה נוספת אודות הפרטים הטכניים יש לפנות לנספחים.

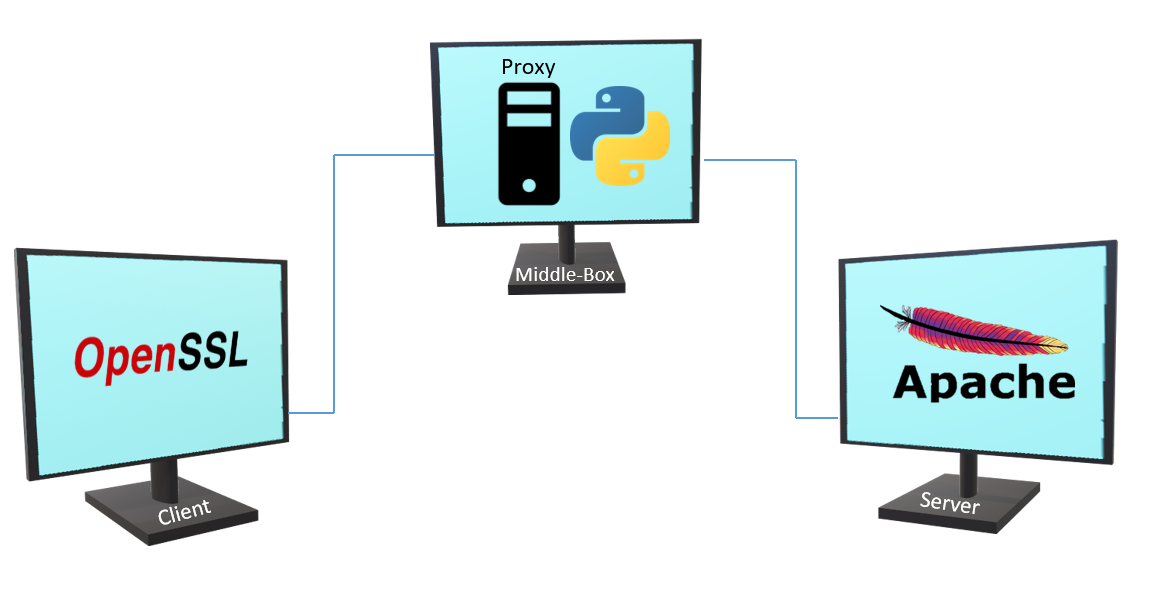
### 5.3.1 תצורת הלקוח

* קובעה כתובת *IP* סטטית השייכת לתת הרשת של הלקוח וה-*middlebox*
* הורץ קוד המממש את פרוטוקול ה-*TLS* לאחר שבוצעו בו שינויים מתאימים

### 5.3.2 תצורת ה-*middlebox*

* קובעה כתובת *IP* סטטית השייכת לתת הרשת של הלקוח וה-*middlebox*
* קובעה כתובת *IP* סטטית השייכת לתת הרשת של השרת וה-*middlebox*
* הורץ קוד *Python* המממש את ה-*proxy* לאחר שבוצעו בו שינויים מתאימים
* הגדרות הניתוב שונו לצורך מעבר תקין של חבילות המידע בין הרשתות

### 5.3.3 תצורת השרת

* קובעה כתובת *IP* סטטית השייכת לתת הרשת של השרת וה-*middlebox*
* הורץ שרת *apache2* (עבורו נוצרה תעודה מתאימה)
* בוצעו שינויים בקובץ ההגדרות של שרת ה-*apache2* לצורך קיבוע חבילת ההצפנה שנבחרה

##### *איור 14: המכונות הוירטואליות ומאפיינן*

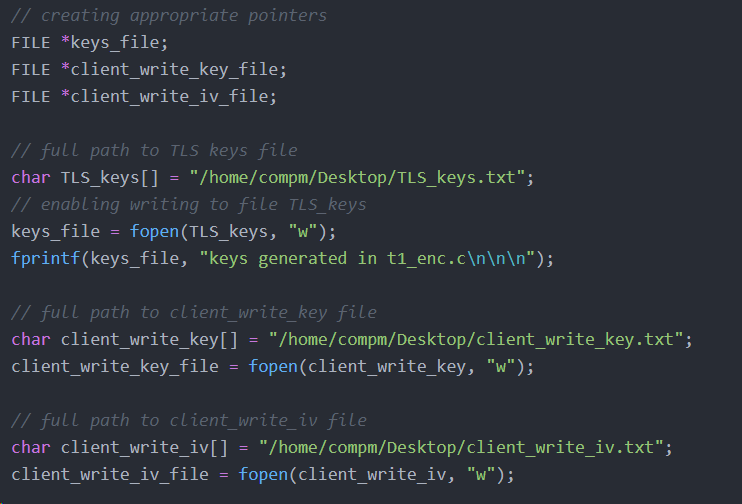
# פרק 6 – השינויים בקוד

## 6.1 התאמת קוד ה-*TLS*

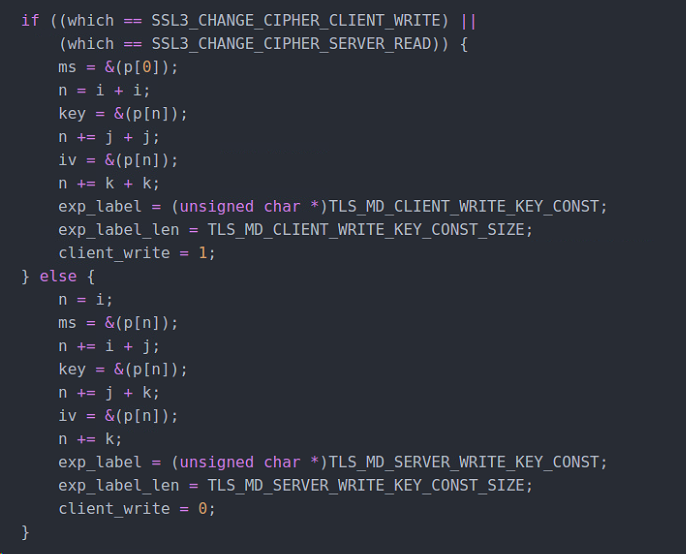
קוד המקור של *OpenSSL*, המממש את פרוטוקול ה-*TLS* וכתוב בשפת *C* דורש שינויים על מנת שנוכל לחלץ מתוכו את מפתחות ההצפנה הסימטריים הנוצרים במהלך לחיצת היד המתבצעת בין שני הצדדים. השינויים בקוד חייבים להיות מבוצעים נקודתית ובמקום הנכון שכן קוד המקור כולל מאות קבצי קוד המכילים עשרות אלפי שורות.

מעקב אחר הלוגיקה של הקוד מוביל אותנו אל הקובץ *t1\_enc.c* הממוקם בספריית *“ssl”.* בקובץ זה, כתובות הפונקציות המתייחסות אל השימוש בפרוטוקול *TLSv1.x* כאשר (כפי שתואר בפרק 2) המשתנה *x* מתאר את גרסת ה-*TLS* בשימוש.

שורות הקוד הבאות אותן נסביר בקצרה, נוספו לפונקציה *tls1\_change\_cipher\_state* שכן זהו השלב בו נוצר ה-*key block* והשלב בו בלוק זה נחתך לטובת המפתחות.

ראשית, נוצרו הקבצים אליהם נרצה לכתוב את המפתחות, וניתן אליהם אישור כתיבה כמתואר באיור הבא

##### *איור 15: קוד ה-TLS - יצירת קבצי המפתחות*

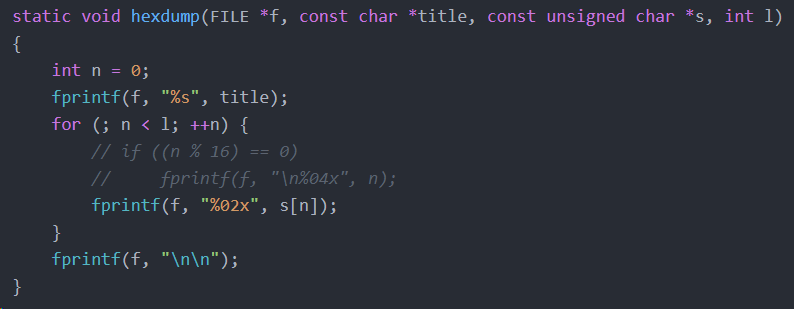
קטע הקוד הנ"ל מתאר את חלוקת המפתחות. נציין כי קטע קוד זה לא שונה וזהה למקור.

##### *איור 16: קוד ה-TLS - חלוקת המפתחות לפי האינדקסים המתאימים*

אם כן, חלוקת המפתחות מתבצעת בצורה הבאה: בלוק המפתחות ‒ *key block* המסומן בעזרת המשתנה *p*  מחולק כך ש-*i* תווים ייכתבו למפתח ה-*MAC* המסומן על ידי *ms* , *j*  תווים ייכתבו למפתח ה-*key* ו-*k* תווים ייכתבו למפתח   
ה-*iv*.   
נבחין כי אם מדובר בכתיבה של הלקוח ‒ *client\_write* או קריאה של השרת ‒ *server\_read,* מפתחות הצפנה זהים, ובאופן דומה, אם מדובר בקריאה של הלקוח ‒ *client\_read* או כתיבה של השרת ‒ *server\_write* מפתחות הצפנה זהים.

לכן, קטע הקוד הבא מבצע כתיבת המפתחות אל תוך הקבצים הנוצרו בתחילת הקוד.

##### *איור 17: קוד ה-TLS - כתיבת המפתחות לקבצים*

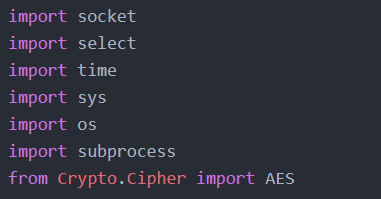
נבחין כי נעשה שימוש בפונקצית מעטפת הנקראת *hexdump*. פונקציה זו נוצרה על מנת להציג את מפתחות ההצפנה בצורה נוחה יותר לשימוש (ייצוג הקסהדצימלי) ומובאת להלן

##### *איור 18: קוד ה-TLS - המרת המפתחות להקסהדצימלי*

## 6.2 התאמת קוד ה-*Proxy*

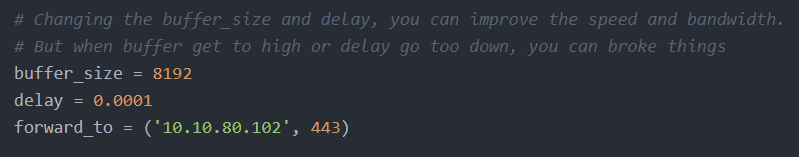
העברת חבילות המידע מרשת אחת לאחרת דורשת ניתוב. משימה זו מבוצעת (בפתרון המוצע) ע"י *proxy*, קוד אשר מורץ על גבי ה-*middlebox* ומאפשר לנו לשלוט על התעבורה המגיעה והיוצאת.   
הרעיון הכללי העומד מאחורי קוד זה, הוא פתיחת שקע ‒ *network socket* עבור זרם נתונים בתקשורת בין תהליכים על גבי רשת המחשבים. שקע זה מאפשר שליטה ובקרה על תעבורת המידע. לכן נוכל, בעזרת המפתחות הסימטריים אשר הלקוח מספק, לצפות בתוכן החבילות מבלי לפצל את התקשורת כפי שנעשה עד היום (כפי שהוסבר בהרחבה בפרק 3).

השינויים שבוצעו בקוד מאפשרים ניטור אחר המידע העובר בין שני הצדדים. נוסף על כך, נציין כי קוד זה כתוב בשפת 3*Python*, ורובו המכריע מבוסס על קוד15 הכתוב בשפת *Python2* ועל כן נאלץ לעבור תרגום.

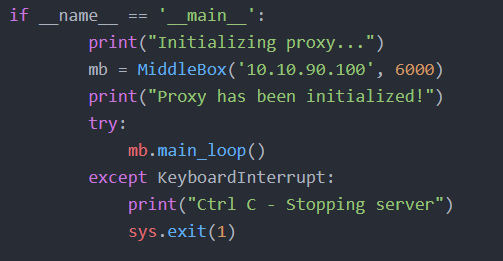
ראשית, יש לייבא את הספריות המתאימות, שכן ללא ספריית הקריפטוגרפיה, לא תתאפשר ההצפנה והפיענוח.

##### *איור 19: קוד ה-proxy - ייבוא הספריות המתאימות*

שנית, הוגדרו כתובת היעד והפורט המתאים להאזנה.



##### *איור 20: קוד ה-proxy - שינוי כתובת היעד*

כמו כן, בעת אתחול הפרוקסי הוגדרו כתובת מקור ופורט להאזנה על גבי ה-*middlebox.*

##### *איור 21: קוד ה-proxy - שינוי כתובת המקור ואתחול ה-proxy*

הוגדרה מחלקה בשם *Decryptor* שמטרתה, בהינתן המפתחות הקריפטוגרפיים המתאימים (מפתחות ה-*session*), לפענח את חבילות המידע העוברות דרך הפרוקסי בדרכן אל היעד. יש לציין כי הקוד לא עושה שימוש במפתח   
ה-*MAC* שכן המימוש שלנו בדק את תוכן החבילות לצורך הפלת תקשורת לא חוקית (הוגדרה כחבילה המכילה מחרוזת מסויימת וקבועה מראש).

המחלקה יוצרת אובייקט קריפטוגרפי מסוג *AES* התואם לבחירת חבילת ההצפנה שביצענו, בעזרתו מתאפשר פענוח (ובמידת הרלוונטיות, גם הצפנה) של חבילת המידע המתקבלת. בכל פעם שנרצה לפענח חבילת מידע נשתמש בפונקציה *decrypt*. חשוב לציין כי הכנסת מפתחות ההצפנה לאובייקט מתבצעת בזמן יצירתו, פעם אחת בלבד. מכאן והלאה, בכל פעם שחבילת מידע תפוענח, האובייקט יידאג להזיז את ה-*IV* בהתאם.

##### *איור 22: קוד ה-proxy - הגדרת המחלקה Decryptor*

כאמור לעיל, יצירת המפתחות אינה מספיקה כשלעצמה, שהרי ה-*middlebox* לא יוכל לפענח את התקשורת המוצפנת בלעדיהם. ועל כן, הלקוח נדרש להעביר את מפתחות ה-*session* אל ה-*middlebox*.   
קטע הקוד הבא מתאר כיצד מתבצעת שליפת המפתחות, קריאתם והמרתם לבתים, ולבסוף, איתחול אובייקט מסוג *decryptor* המשמש לפענוח התעבורה המוצפנת (כפי שהוסבר לעיל).

נרצה לציין שהעברת המפתחות מתבצעת בעזרת תסריטים (*scripts*) קצרים שכתבנו המשתמשים   
ב-*SSH secure file transfer.*



##### *איור 23: קוד ה-proxy - שליפת המפתחות מהלקוח*

המידע הנקרא בצורתו המוצפנת עובר תהליך פירסור (*parsing*) בו הוא מומר לתווים הקסהדצימלים אליהם נדרשת התייחסות מתאימה.

* במידה וההודעה המתקבלת מתחילה (שלושת התווים הראשונים) במחרוזת "16", הרי שמדובר בהודעה מסוג *“handshake message”.* יש להבחין בין ההודעות הללו שכן פירושה של אחת מהן הוא שמפתחות ה-*session* נוצרו ומעתה התוכן המועבר מוצפן. לכן, זהו השלב בו המפתחות נשלפים.
* במידה וההודעה המתקבלת מתחילה במחרוזת "17" הרי שמדובר בהודעה מסוג *“application data”*. כאמור בפרק 2, הודעות אלה נשלחות רק לאחר תהליך לחיצת היד והן למעשה המידע בצורתו המוצפנת הנשלח על ידי הצד השולח.



##### *איור 24: קוד ה-proxy - פירסור חבילות המידע*

# פרק 7 – סיכום ורעיונות להמשך

פרוייקט זה מציע דרך ממשית לשילוב צד נוסף (אחד או יותר), בתקשורת מאובטחת ברשת המבוססת על פרוטוקול *TLS*, פרוטוקול שבמקור יועד עבור שני צדדים בלבד. הפתרון המוצע עושה זאת מבלי הצורך במפתחו הפרטי של השרת, ומחייב את ה-*middlebox* (הצד הנוסף) להזדהות בפני הלקוח טרם קבלת מפתחות השיחה. בכך, הפתרון מגביר את האבטחה ברשת.

הקושי העיקרי בפרוייקט שכזה הוא פענוח חבילת המידע הראשונה, שכן פענוח זה טומן בחובו את שליפת המפתחות הקריפטוגרפיים הנכונים, תוך שימוש בשורות הקוד הנכונות, במיקום הנכון ובזמן הנכון. נוסף על כך, הקמת תשתית מערכת הכוללת שלוש מכונות וירטואליות ביניהן קיימת תקשורת ישירה (פרט לכזו בין הלקוח והשרת) פותחת צוהר לקשיים טכניים רבים המהווים מכשולים לא פשוטים בפני עצמם.

נדגיש בשנית, המימוש הנוכחי לפתרון המוצע אינו שלם במובן שאינו מקיים את כלל הדרישות התיאורטיות. לדוגמה, המימוש אינו כולל את הזדהותו של הצד הנוסף בפני הלקוח לשם קבלת מפתחות השיחה הסימטריים ואינו כולל את בקשתו המפורשת לקבלתם. לפיכך, פיתוח נוסף של הפתרון המוצע יהיה יישום ההזדהות של ה-*middlebox* בפני הלקוח ובקשתו עבור קבלת מפתחות השיחה.

כמובן שאין להתעלם מכך שקיים סיכון אבטחתי, גם אם מזערי, בכך שמפתחות השיחה נמצאים אצל צד שלישי ועל כן נדרשת חומרה או תוכנה מאובטחת בה יישמרו המפתחות. לכן, כיוון מעניין בו פרוייקט זה יכול להתפתח הוא שמירת המפתחות ואבטחתם מפני תוקפים פוטנציאליים.

# טבלת איורים

[*איור 1: תיאור פעולת אלגוריתם Diffie-Hellman* 7](#_Toc529668911)

[*איור 2: תיאור פעולת אלגוריתם CBC* 9](#_Toc529668912)

[*איור 3: דוגמה לחולשה באלגוריתם EBC לעומת CBC* 9](#_Toc529668913)

[*איור 4: דוגמה לחבילת צופן* 11](#_Toc529668914)

[*איור 5: משמעות השדות שבחבילת הצופן* 11](#_Toc529668915)

[*איור 6: פעולת החתימה ולאחריה ההצפנה* 12](#_Toc529668916)

[*איור 7: סכימה לתיאור לחיצת היד של פרוטוקול ה-TLS* 13](#_Toc529668917)

[*איור 8: חלוקת הבלוק למפתחות* 16](#_Toc529668918)

[*איור 9: מבנה הרשת הקיימת* 17](#_Toc529668919)

[*איור 10: מבנה הרשת לפי הפתרון המוצע* 19](#_Toc529668920)

[*איור 11: הגדרת ה-NAT* 22](#_Toc529668921)

[*איור 12: הגדרת intnet\_Client\_MiddleBox* 23](#_Toc529668922)

[*איור 12: הגדרת intnet\_Client\_MiddleBox* 24](#_Toc529668923)

[*איור 13: טופולוגיית הרשת הממומשת* 24](#_Toc529668924)

[*איור 14: המכונות הוירטואליות ומאפיינן* 26](#_Toc529668925)

[*איור 15: קוד ה-TLS - יצירת קבצי המפתחות* 27](#_Toc529668926)

[*איור 16: קוד ה-TLS - חלוקת המפתחות לפי האינדקסים המתאימים* 27](#_Toc529668927)

[*איור 17: קוד ה-TLS - כתיבת המפתחות לקבצים* 28](#_Toc529668928)

[*איור 18: קוד ה-TLS - המרת המפתחות להקסהדצימלי* 28](#_Toc529668929)

[*איור 19: קוד ה-proxy - ייבוא הספריות המתאימות* 29](#_Toc529668930)

[*איור 20: קוד ה-proxy - שינוי כתובת היעד* 30](#_Toc529668931)

[*איור 21: קוד ה-proxy - שינוי כתובת המקור ואתחול ה-proxy* 30](#_Toc529668932)

[*איור 22: קוד ה-proxy - הגדרת המחלקה Decryptor* 31](#_Toc529668933)

[*איור 23: קוד ה-proxy - שליפת המפתחות מהלקוח* 31](#_Toc529668934)

[*איור 24: קוד ה-proxy - פירסור חבילות המידע* 33](#_Toc529668935)

# ביבליוגרפיה

1. <https://en.wikipedia.org/wiki/Public-key_cryptography>
2. <https://www.techopedia.com/definition/24749/cryptographic-key>
3. <https://en.wikipedia.org/wiki/Diffie–Hellman_key_exchange>
4. <https://en.wikipedia.org/wiki/Block_cipher>
5. <https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation>
6. <https://hpbn.co/transport-layer-security-tls/>
7. <https://en.wikipedia.org/wiki/Message_authentication_code>
8. <https://www.cryptologie.net/article/340/tls2-pre-master-secrets-and-master-secrets/>
9. <http://kodu.ut.ee/~b04866/tls.pdf>
10. <https://davidtnaylor.com/mbTLS.pdf>
11. <http://tce.webee.eedev.technion.ac.il/wp-content/uploads/sites/8/2016/09/tce-2016-day2.pdf>
12. <https://cyber-defense.sans.org/resources/papers/gsec/ssl-tls-hood-126976>
13. <https://tools.ietf.org/html/rfc5246>
14. <https://en.wikipedia.org/wiki/Transport_Layer_Security>
15. <http://voorloopnul.com/blog/a-python-proxy-in-less-than-100-lines-of-code/>

# *נספחים*

## נספח א': בדיקת תקשורת בין המכונות הוירטואליות

לפני שילוב הפרוקסי, נרצה לוודא שהתקשורת בין המכונות תקינה

הקמת ה- *MiddleBox*:

* קינפוג כרטיס הרשת בצד הלקוח לIP סטטי: 10.10.90.101
* קינפוג כרטיס הרשת בצד השרת לIP סטטי: 10.10.80.102

בדיקת פינגים בין המכונות לוודא שהתקשורת עובדת.

הגדרות הרשת:

* הגדרת התקשורת לעבור אך ורק דרך ה- *MiddleBox* ולא ישירות מהלקוח לשרת בעזרת הפקודה:

*sudo route add -net 10.10.90.0 255.255.255.0 gw 10.10.90.101*

לאחר ביצוע פקודה זו נוספת רשומה לטבלת הניתוב של המחשב המראה שבכל מקרה בו הוא רוצה להעביר הודעה למחשב כלשהו ברשת 10.10.90.0 הוא חייב לעבור קודם כל דרך המחשב 10.10.90.101.  
את שורות הניתוב ניתן לראות בעזרת הפקודה:

*route -n*

למקרה הצורך, הפקודה שמוחקת את הרשומה הזאת היא:

*sudo route del -net 10.10.90.0 netmask 255.255.255.0*

* על מנת שהתקשורת תעבוד בתצורה הנוכחית נצטרך לבצע *IPv4* Forwarding.  
  בכדי לבדוק האם *IPv4 Forwarding* מאופשר על המכונה יש לבצע את הפקודה הבאה:

*cat /proc/sys/net/ipv4/ip\_forward*

אם מוחזר 0 זה לא מאופשר, אחרת מאופשר.

אם זה לא מאופשר יש שתי אפשרויות:

* + - לאפשר זמנית באמצעות הפקודה הבאה:

*syscl -w net.ipv4.ip\_forward=1*

* + - לאפשר לצמיתות ע"י מחיקת הערה בקובץ *sysctl.conf* בשורה 28  
      לא ניתן לכתוב לקובץ סתם כך, צריך להשתמש ב-*sudo* :

*sudo gedit /etc/sysctl.conf*

כדי להחיל את השינויים יש להריץ:

*sudo sysctl -p /etc/sysctl.conf*

או

*/etc/init.d/procps restart*

כעת נבחין שהתקשורת תעבוד כמצופה.

## נספח ב': בדיקת תקשורת בין המכונות הוירטואליות דרך שרת הפרוקסי

עלינו לקנפג את הרשתות בצורה כזו שכל חבילה שעוברת בין הלקוח לשרת, תעבור דרך הפרוקסי שרץ על ה-*mb*.

לצורך כך, מפורטת הגדרת הרשת בצורה הבאה:

*middlebox – enp0s9 – 08:00:27:d1:6c:01 – 10.10.80.100*

*middlebox – enp0s8 – 08:00:27:6d:0d:09 – 10.10.90.100*

*client – enp0s8 – 08:00:27:9d:31:00 – 10.10.90.101*

*server – enp0s9 – 08:00:27:5f:77:14 – 10.10.80.102*

* לאחר הקמת 2 הרשתות הפנימיות בין המכונות הוירטואליות השונות (פעולה המבוצעת בהגדרות המכונה הוירטואלית – *virtualbox*), עלינו להגדיר את כתובות ה-*IP*, מסכת הרשת והנתב הדיפולטי בצורה ידנית.

כעת עלינו להריץ את הפקודות הבאות **על ה-*middlebox*** על מנת שהתעבורה תעבור דרך הפרוקסי.

הפקודה הבאה תעביר את התעבורה שמגיעה מהרשת *enp0s8* לפורט *6000* לו מאזין הפרוקסי.

*sudo iptables -t nat -A PREROUTING -i enp0s8 -p tcp -j REDIRECT --to-port 6000*

**הערה**: הפורט הנבחר (*6000*) הינו פורט רנדומלי.

הפקודה הבאה תעביר את התעבורה שמגיעה מהפרוקסי לרשת *enp0s9* לה מאזין השרת

*sudo iptables -t nat -A POSTROUTING -o enp0s9 -p tcp*

פילטר ב-*Wireshark* המאפשר לראות את חבילות ה-*TLS* בלבד

*ssl.record.version == 0x0301 || ssl.record.version == 0x0303*

הצגת שרשראות (*chains*) של טבלאות *nat*

*sudo iptables -t nat -L -nv*

מחיקת שרשרות של טבלאות *nat*

*sudo iptables -t nat -F*

## נספח ג': קיבוע חבילת ההצפנה על גבי שרת ה-*Apache*

יש לגשת לנתיב הבא

*/etc/apache2/mods-enabled/*

שם, יש לפתוח את הקובץ *“ssl.conf”* ושם יש לשנות את השורה

*“SSLCipherSuite HIGH:!aNULL”*

לחבילת ההצפנה המבוקשת. לדוגמה,

*“SSLCipherSuite DHE-RSA-AES128-SHA256”*

## נספח ד': פקודות לקימפול קוד ה-*TLS*

הוראות קימפול רשמיות בכניסה למדריך הבא (יש להשתמש בהרשאות מתקדמות):

*Desktop\TLS\_Source\_Code\openssl-1.0.2n\INSTALL*

שלבי הקמפול:

* כניסה למדריך הבא *Desktop\TLS\_Source\_Code\openssl-1.0.2n* ופתיחת הטרמינל במדריך זה
* הכנסת הפקודה הבאות לפי הסדר:
  1. *sudo ./config --openssldir=/usr/local/apps/openssl-1.0.2n*
  2. *sudo make*
  3. *sudo make install*

## נספח ה': הוספת ספריית הקריפטוגרפיה של *Python*

התקנת ספריית הקריפטוגרפיה של *Python3*מתבצעת בעזרת השורה הבאה:

sudo pip3 install cryptography

1. <https://en.wikipedia.org/wiki/Public-key_cryptography>

   <https://www.techopedia.com/definition/24749/cryptographic-key> [↑](#footnote-ref-1)
2. <https://www.hamichlol.org.il/%D7%A4%D7%A8%D7%95%D7%98%D7%95%D7%A7%D7%95%D7%9C_%D7%93%D7%99%D7%A4%D7%99-%D7%94%D7%9C%D7%9E%D7%9F> [↑](#footnote-ref-2)
3. https://en.wikipedia.org/wiki/Block\_cipher [↑](#footnote-ref-3)
4. https://en.wikipedia.org/wiki/Block\_cipher\_mode\_of\_operation [↑](#footnote-ref-4)
5. https://hpbn.co/transport-layer-security-tls/ [↑](#footnote-ref-5)
6. <https://hpbn.co/transport-layer-security-tls/> [↑](#footnote-ref-6)
7. https://hpbn.co/transport-layer-security-tls/ [↑](#footnote-ref-7)
8. <https://www.cryptologie.net/article/340/tls2-pre-master-secrets-and-master-secrets/> [↑](#footnote-ref-8)