

**ספר פרויקט**

תשפ"ה 2024

**TOR NETWORK**

****

**צוות:** תמר עמרם ויונתן רודריגז

**כיתה:** 19 מוקד כנרת

**ראש צוות:** ניסן פיכמן

**מנטור:** ניסן פיכמן

1. 
2. **פרק 1: יזום**
   * 1. **:תיאור כללי**

הפרויקט עוסק בניתוב ואבטחת רשתות באמצעות תור (The Onion Router)  
המתמקד בתקשורת פרטית מוצפנת באינטרנט. הנתונים "יעטפו" בשכבות הצפנה בדומה לשכבות של הבצל וישלחו דרך מספר צמתים המכונים "dockers" (קונטיינרים). תהיה התמקדות בפיתוח והטמעה של מנגנון ניתוב, כולל הצפנה רב-שכבתית, תוך שימוש בהצפנה אסימטרית להחלפת מפתחות.  
כגון RSA הצפנה סימטרית, וכגון AES בהצפנת הנתונים. הפרויקט יעסוק גם באבטחת רשת ובטיפול באתגרים כמו מתקפות Replay ו-Man-in-the-Middle באמצעות פרוטוקולים חזקים ואמצעי הגנה מתקדמים.

**מטרת הפרויקט:**

כיום, אנשים רבים בכל רחבי העולם משתמשים באופן יומיומי ברשת האינטרנט ובעקבות כך חשופים לסכנות ולמידע אישי שלהם ועל פעולותיהם ברשת אשר נאסף ומשמש לדברים שונים או נמכר לחברות צד שלישי המנתחות את המידע לשימושן. אנו מזהים את חשיבות הפרטיות בעולם הגלובלי והפתוח בו אנחנו חיים ולכן מטרת הפרויקט הינה לממש רשת אשר מאפשרת לפעול בצורה פרטית ובטוחה, ובה רק לשני קצוות, המקור והיעד, יש גישה למידע עצמו.

**קהל היעד:**

קהל היעד שלנו הוא קהל יעד רחב ומגוון של אנשים. כל אדם המעוניין לפעול ברשת בצורה מאובטחת ופרטית לכל צרכיו השונים מבלי לדאוג שהמידע שלו יאסף יכול להשתמש בתור לדוגמה אנשים פרטיים שרוצים לצרוך תוכן, להפעיל שרתים ולקבל מידע באופן בטוח מקצה לקצה.

* 1. **פתרונות קיימים:**  
     **--------------------------------------------------------------------------------------------------------------**  
     **1. Tor Browser**
     1. Tor Browser הוא הפרויקט המוכר ביותר, מבוסס על רשת (TOR (The Onion Router, המיועד לאפשר למשתמשים לגלוש באינטרנט באופן אנונימי. הדפדפן פועל על ידי ניתוב חבילות נתונים דרך שכבות של הצפנה בצמתים רבים (צמתים המכונים "onion routers"), כך שהמיקום והזהות של המשתמש מוסתרים. Tor Browser מאפשר גלישה באתרים רגילים וגם גישה לאתרים עם סיומת .onion, המהווים חלק מהאינטרנט האפל (Dark Web).

**מאפיינים עיקריים**:

* + 1. o ניתוב חבילות דרך מספר צמתים מבוזרים לשמירה על פרטיות.
    2. o גישה לאתרי אינטרנט אנונימיים(Hidden Services)עם סיומת .onion.
    3. o הגנה מפני מעקב, ניתוח תעבורה, ומתקפות Man-in-the-Middle.
    4. o נגישות גבוהה, עם קוד פתוח ותמיכה בקהילות גלובליות.

**קישורים רלוונטיים -**<https://www.torproject.org/download/>

* 1. **--------------------------------------------------------------------------------------------------------------**
  2. **2. Freenet**

Freenet מתמקד ביצירת רשת מבוזרת לשיתוף תכנים באופן אנונימי ומאובטח. המשתמשים מאחסנים ומורידים תכנים תוך הצפנתם ופיזורם ברשת, בדומה לרעיון של TOR.

**מאפיינים עיקריים**:

o רשת לשיתוף תכנים מבוזרת.

o הצפנה חזקה ושמירה על פרטיות המשתמשים.

o פלטפורמה לשמירה על חופש הביטוי והגנה מצנזורה.

**קישורים רלוונטיים -**

<https://freenet.org/>

* 1. **--------------------------------------------------------------------------------------------------------------**
  2. **3. GNUnet**
     + 1. GNUnet הוא פרויקט נוסף המספק שירותי רשת אנונימיים מבוססי ניתוב חבילות מוצפן. הוא מתמקד בשירותי שיתוף קבצים ואנונימיות של המשתמשים.

**מאפיינים עיקריים**:

o ניתוב חבילות אנונימי ושירותי רשת מבוזרים.

o אבטחת רשת חזקה ושמירה על פרטיות.

**קישורים רלוונטיים -**

<https://www.gnunet.org/en/>

* 1. **--------------------------------------------------------------------------------------------------------------**

**ליבה טכנולוגית:**  
 הליבה הטכנולוגית בפרויקט הינה מעבר חבילה מנקודה אחת לאחרת (מקור ויעד) בצורה מוצפנת.

**העברת חבילות**:  
העברת חבילות ברשת TOR מתבצעת על ידי צמתים (מחשבים של משתמשים ברשת).  
בדומה לרכיבי רשת רגילים שמעבירים חבילות בפלטפורמה רגילה, גם ברשת TOR מתקיימת העברת חבילות בין צמתים.

**הצפנת החבילות**:  
הצפנת החבילות ברשת TOR נעשית באופן שונה מהצפנה רגילה, שבה הקצה השולח מצפין והקצה המקבל מפענח. ברשת, הקצה השולח מצפין את החבילה במספר שכבות לפי מספר הצמתים שהיא תעבור בהם, בנוסף להצפנה הרגילה. ההצפנה לפי מספר הצמתים מתבצעת באמצעות הצפנה אסימטרית, הכוללת שני סוגי מפתחות. המפתח הראשון, המשמש להצפנה, נגיש לכולם ומשמש את השולח. המפתח השני, המשמש לפענוח, נמצא רק בידי הקצה המקבל, ורק הוא יכול לפענח את ההצפנה.

**טכנולוגיות עיקריות:**

**שפות התכנות שנשתמש בהן**: c++\c  
שפות אלו הינן שפות תכנות מאוד נפוצות הנחשבות לשפות low level ובהן התמקדנו במשך השנתיים שלנו בתוכנית מגשימים. (בחרנו בהן מכיוון שאנחנו מרגישים שהשפות הללו יאפשרו לנו להתעמק בפרויקט בנושא רשתות ברמה הרצויה)

**חיבור nodes והעברת המידע ביניהם באופן מחושב במסלול אופטימלי**:  
את יצירת המסלול אנחנו נעשה על ידי שיתוף מידע מהשרת דירקטורי על הצמתים הקיימים שנמצאים ברשת ולאחר מכן המסלול יוחלט על ידי אלגוריתם פשוט לפי אופי הרשת בצד לקוח. הטכנולוגיה הזו היא המשמעותית ביותר ברשת הטור שהוא הפרויקט שלנו.

**הצפנת השכבות על ידי AES** :  
הצפנת AES היא אלגוריתם הצפנה שמצפין באמצעות מפתח המחולק לארבעה מפתחות  
שכל אחד מהם אחראי על דרך ריבוב שונה בהצפנה הכוללת.  
את הצפנה אנחנו נעשה על ידי האלגוריתם AES, אלגוריתם הצפנה אסימטרי אשר נצפין דרכו אחרי בקשת המפתחות הפומביים מהשרת ולאחר מכן נצפין את ההודעה בכמות המפתחות (כמות הצמתים של המסלול המחושב), על כל שכבה מוצפנת נוסיף headers המכיל את הפרטים של הצומת הבא בחרנו בה מכיוון שכיום היא נחשבת לאלגוריתם ההצפנה הנפוץ ביותר והבטוח ביותר המתאים כוונותינו.

**אתגרים טכנולוגיים ומקורות מידע:** בעת העבודה על הפרויקט אנחנו עשויים להיתקל באתגרים טכנולוגיים שונים כדוגמת:

**--------------------------------------------------------------------------------------------------------------**

**1. ניהול זיכרון ומבני נתונים:**

בשפות כמו C ו-C++, ניהול זיכרון הוא אתגר קריטי. השפות נותנות גישה ישירה לזיכרון, לכן נצטרך להתמודד עם דליפות זיכרון, ניהול נכון של מצביעים, ושימוש יעיל במבני נתונים כמו רשימות מקושרות ותורים (queues) כדי לנהל את החבילות בצורה אפקטיבית.

* **אתגר:**
  + שמירה על יעילות בניהול זיכרון בעבודה עם חבילות גדולות ומספר צמתים בו-זמנית.
  + מניעת דליפות זיכרון וניצול משאבים נכון.
* **מקורות מידע:**
  + Memory Management in C/C++
  + תיעוד רשמי של C++: C++ Reference

**--------------------------------------------------------------------------------------------------------------**

**2. הצפנה רב-שכבתית (Asymmetric and Symmetric Encryption)**

TOR משתמש בהצפנה רב-שכבתית כדי להבטיח שכל צומת רק מפענח את השכבה המתאימה לו, ומשאיר את השאר מוצפנות. הפעלת הצפנה א-סימטרית (כמו RSA) והצפנה סימטרית (כמו AES) בשפת C/C++ מצריכה הבנה מעמיקה במבני ההצפנה ובספריות מתאימות.

* **אתגר:**
  + יישום פרוטוקולי הצפנה כמו RSA ו-AES בצורה יעילה ומאובטחת.
  + הבטחת נכונות ההצפנה בכל שלבי הניתוב בין הצמתים.
* **מקורות מידע:**
  + ספריות הצפנה פופולריות - [OpenSSL](https://www.openssl.org/) , [libsodium](https://libsodium.org/).
  + תיעוד בנושא RSA ו-AES: RSA Algorithm in C++ ו-AES Encryption

**--------------------------------------------------------------------------------------------------------------**

**3. תקשורת ברשת (Networking)**

ניהול תקשורת בין הצמתים יכול להיות מאתגר, נצטרך לתכנן כיצד להפעיל TCP/IP וליישם את ניהול החבילות בצורה נכונה, תוך שמירה על ביצועים ואמינות.

* **אתגר:**
  + יישום פרוטוקול תקשורת (כמו TCP) בצורה יעילה ומאובטחת.
  + טיפול בחבילות שאבדו, ניהול עומסים, ומניעת מתקפות כמו Replay.
* **מקורות מידע:**
  + ספריית תקשורת: Boost.Asio (ספריית רשת פופולרית ל-C++).
  + Beej’s Guide to Network Programming - מדריך מעמיק בנושא תקשורת ברשת ב-C.
  1. **--------------------------------------------------------------------------------------------------------------**

**4. אופטימיזציה וביצועים**

בניית רשת TOR דורשת אופטימיזציה של קוד לניהול תעבורת נתונים בצורה מהירה ואפקטיבית. צריך לוודא שהקוד לא מכביד על ה-CPU ושניצול הזיכרון יעיל גם בעומסים גבוהים.

* **אתגר:**
  + אופטימיזציה של תהליך הניתוב וההצפנה כדי למנוע עיכובים.
  + שיפור מהירות העברת חבילות בין הצמתים.
* **מקורות מידע:**
  + Performance Optimization in C++ - מדריכים לאופטימיזציה.
  + Efficient Algorithms in C++
  1. **--------------------------------------------------------------------------------------------------------------**

**5. מניעת מתקפות והבטחת אבטחת מידע**

TOR פגיע למתקפות כמו מתקפות Replay ו-Man-in-the-Middle, לכן נטמיע פרוטוקולים שימנעו את התקפות אלה ונבטיח שהמידע עובר בצורה מאובטחת בין הצמתים.

* **אתגר:**
  + מניעת מתקפות Replay על ידי יישום פרוטוקולי אבטחה מתאימים.
  + אבטחת הצפנה לכל אורך תהליך הניתוב בין הצמתים.
* **מקורות מידע:**
  + Secure Coding in C/C++
  + MITM Prevention Techniques

מניעת ניתוח תעבורה (Traffic Analysis)

אחד האתגרים הגדולים ברשת TOR הוא מניעת ניתוח תעבורה (Traffic Analysis), שבו תוקפים מנסים ללמוד על דפוסי תנועה כדי לזהות משתמשים ברשת. יש למצוא דרכים להסתיר את דפוסי התקשורת כך שלא ניתן יהיה להסיק מזה מסקנות על זהות המשתמשים.

* **אתגר:**
  + יצירת מנגנונים שמטשטשים את דפוסי התעבורה כך שלא ניתן לנתחם.
* **מקורות מידע:**
  + [Research Papers on Traffic Analysis in TOR](https://arxiv.org/list/cs.CR/recent) - מאמרים ומחקרים על TOR ואנונימיות.

**--------------------------------------------------------------------------------------------------------------**

**דרישות חומרה:**  
מחשב

1. 
2. **פרק 2: אפיון**
   1. **פיצ'רים ותהליכים עיקריים:**

**—-----------------------------------------------------------------------------------**

**שכבות ההצפנה -**

החבילה שהמשתמש שולח עוברת תהליך הצפנה רב-שלבי כדי להבטיח שהמידע לא יהיה קריא לרכיבים שמקבלים את החבילה בדרך. התהליך מתבצע כך:

1. **בקשה לשרת הדירקטור**: המערכת בצד הלקוח שולחת בקשה לשרת הדירקטור לקבל רשימה של מחשבים (צמתים) ברשת והמפתחות הפומביים השייכים להם.
2. **בחירת סדר הצמתים**: לאחר קבלת הנתונים, המערכת בוחרת, על פי אלגוריתם מוסכם מראש או רנדומלי, את סדר הצמתים שהחבילה תעבור בהם במהלך המסע שלה ברשת.
3. **הצפנת החבילה**: המערכת מתחילה [להצפין את החבילה](#s39lvjsfht8) עבור כל צומת ברשת, מהסוף להתחלה. כלומר, החבילה מוצפנת קודם כל עם המפתח הפומבי של הצומת האחרון, אחר כך שלפניו, וכן הלאה עד להצפנה עם המפתח של הצומת הראשון.
4. **הוספת headers**: בכל שכבת הצפנה נוספת, המערכת מכניסה headers (כותרות) לא מוצפנים שמכילים את הפרטים של הצומת הבא שהחבילה אמורה לעבור בו. הכותרות הללו מאפשרות לצומת לדעת לאן עליו להעביר את החבילה, מבלי לדעת את התוכן המוצפן שבתוכה.

כך מובטח שכל צומת יוכל לראות רק את השכבה שלו והצומת הבא, מבלי לדעת את המידע המלא או את שאר הצמתים ברשת.

**קלט -** שכבה המכילה את ה data וheader המראה את החוליה הבאה בשרשרת

**פלט -** השכבה לאחר ההצפנה שתשלח בחבילה המוצפנת בשכבות דרך הנתיב ליעד.

**הסבר קוד**   
 הלקוח מבקש את קובץ המפתחות מהשרת ולאחר שהוא מקבל את זה   
 הוא שולח את המפתחות לפונקציה SelectRoute ואז הפונקציה משתמשת במחלקה   
 OnionManager ובפונקציה EncryptLayer   
 לכל אחד מהמפתחות עוד שהוא מסיים את המפתחות ובכך מצפין את החבילה בשכבות

—-----------------------------------------------------

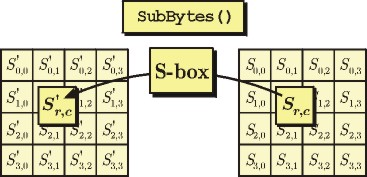
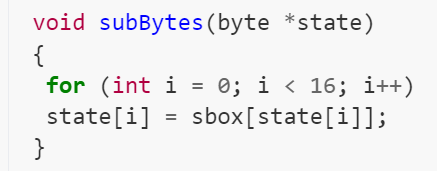
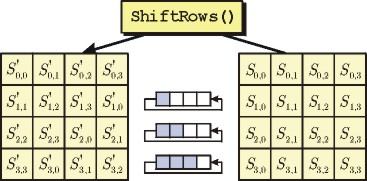
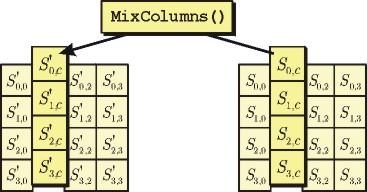
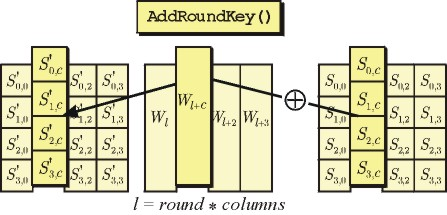
**הצפנת AES -**המערכת תוכל להצפין את חבילות על ידי הצפנת בלוקים aes   
הצפנת aes היא הצפנה אסימטרית המשתמשת במפתח מרובע לפענוח החבילה   
ההצפנה כמו שציינו היא בעצם ארבעה הצפנות נפרדות המופעלות על החבילה

**קלט -** טקסט \ בינארי שמוכנס על ידי המשתמש

**פלט -** בלוק של 256 סיביות קבוע המכיל את ה טקסט \ בינארי

**הסבר על ההצפנה - (ההצפנה הראשית)**

הצפנת AES היא הצפנת בלוקים המשתמשת ברעיון של הצפנת בלוקים, כלומר היא מפעילה אלגוריתם מתמטי המערבב גריד (טבלה). עקרון זה עובד בצורה הבאה:

1. **חלוקת הטקסט המקורי לבלוקים**: תחילה, הטקסט המקורי מחולק לבלוקים בגודל קבוע של 128 ביטים כל אחד. אם הטקסט אינו מתחלק במדויק, הוא ממולא בערכים מתאימים כדי להגיע לגודל הנדרש.
2. **שלבי ההצפנה**: כל בלוק עובר מספר שלבים של ערבוב ושינוי, כאשר כל שלב מורכב מסדרה של פעולות מתמטיות על הגריד. התהליך כולל את השלבים הבאים:
   * **SubBytes**: בכל שלב זה, כל בייט בטבלה מוחלף בערך אחר לפי טבלת חיפוש קבוע הנקראת S-box.  
       
     
   * **ShiftRows**: בשלב זה, השורות בטבלה מוזזות בכמות קבועה של מקומות לשמאל, כך שנוצרת ערבוב נוסף.  
     
   * **MixColumns**: כל עמודה בטבלה עוברת שינוי לפי פעולה מתמטית מסוימת שמוסיפה ערבוב נוסף לבלוקים.  
     
   * **AddRoundKey**: לבסוף, מפתח הצפנה נוסף לכל בלוק בטבלה בעזרת פעולת XOR.  
     
3. **חזרות מרובות**: תהליך ההצפנה מורכב ממספר סבבים (rounds), כאשר כל סבב כולל את כל השלבים הנ"ל. מספר הסבבים נקבע לפי אורך המפתח: 10 סבבים עבור מפתח של 128 ביטים, 12 סבבים עבור מפתח של 192 ביטים, ו-14 סבבים עבור מפתח של 256 ביטים.
4. **פענוח**: תהליך הפענוח של AES הוא ההיפוך המדויק של תהליך ההצפנה, כאשר סדר השלבים והפעולות מתבצע בצורה הפוכה כדי לשחזר את הטקסט המקורי מהבלוקים המוצפנים.

**הצפנות נוספות**   
 - sha2

sha 256 - מייצר HASH של 256 סיביות

פונקציית גיבוב קריפטוגרפית המספקת ערך בגודל קבוע מקלטים באורכים שונים.

הht מתוכננת כך שבלתי אפשרי מבחינה חישובית למצוא שני ערכים שונים המייצרים את אותו הערך. בנוסף, הוא עמיד בפני מתקפות כמו מתקפות קדם תמונה( מציאת קלט שניתן לגבב לפלט נתון) או מתקפות קדם תמונה שניות (מציאת קלט אחר שניתן לגבב לאותו הפלט כמו הקלט הנתון).   
—-----------------------------------------------------

**הצמתים וחיבורם לרשת -**

הצמתים במערכת מתחברים לרשת דרך שרת הדירקטורי, שמנהל את החיבור ומספק את המפתחות הנחוצים לצמתים כדי להשתתף בתהליך ההצפנה. התהליך מתבצע באופן הבא:

1. **בקשת התחברות לשרת הדירקטור**: הצומת שולח בקשה לשרת הדירקטור על מנת להתחבר לרשת. הבקשה כוללת פרטים ראשוניים של הצומת שמבקשת אישור להתחבר.
2. **החזרת מפתח חיצוני**: שרת הדירקטור שולח לצומת מפתח חיצוני (למשל, מפתח ציבורי) שנועד לאמת את החיבור ולהתחיל את תהליך החלפת המפתחות.
3. **שליחת מפתח מהצומת**: לאחר קבלת המפתח החיצוני משרת הדירקטור, הצומת מייצר מפתח משלו ושולח אותו בחזרה לשרת הדירקטור. זהו המפתח שהצומת ישתמש בו כדי לתקשר באופן מאובטח עם שאר הרשת.
4. **עדכון רשימת המפתחות בשרת הדירקטור**: שרת הדירקטור מקבל את המפתח שהצומת יצר, ומעדכן את [רשימת המפתחות](#o4hygwrl36sr) של כל הצמתים ברשת. כך, כל צומת ברשת יכול לקבל את המפתח הציבורי של צמתים אחרים על מנת לבצע את תהליך ההצפנה והתקשורת בצורה מאובטחת.
5. **אישור חיבור והתחלת שימוש ברשת**: לאחר עדכון רשימת המפתחות, שרת הדירקטור נותן לצומת אישור להתחבר לרשת ולהתחיל להשתמש בה. הצומת יכול כעת לשלוח ולקבל חבילות מוצפנות דרך הרשת בצורה מאובטחת.  
     
     
   רשימת המפתחות:  
    רשימת המפתחות תיהיה [בסיס נתונים](#hp6yojou34ci) אשר ישב אצל השרת דירקטורי   
    ומשימתו היא לאכסן את המפתחות הפומביים של כל הצמתים שנמצאים ברשת באותו הזמן

**קלט -** בקשת התחברות לשרת והתחלת ההתחברות לרשת

**פלט -** צומת מתפקדת ברשת

**הסבר קוד**

במערכת המבוססת על **Onion Routing**, תהליך חיבור הצמתים לרשת מתבצע באופן הבא:

1. **בקשת התחברות לשרת הדירקטור**: הצומת (Node) שולח בקשה לשרת הדירקטור כדי להתחבר לרשת. בקשה זו מכילה מידע ראשוני על הצומת, כגון ה-IP שלו, פורט התקשורת, ומפתח ציבורי שניתן לשתף עם שאר הצמתים.
2. **החזרת מפתח חיצוני**: שרת הדירקטור מקבל את בקשת ההתחברות, ומחזיר לצומת מפתח ציבורי (מפתח ציבורי של השרת עצמו או מפתח הצפנה זמני), שבו הצומת ישתמש כדי להתחיל בתהליך החלפת המפתחות.
3. **שליחת מפתח מהצומת**: לאחר קבלת המפתח החיצוני מהשרת, הצומת יוצר מפתח משלו (מפתח ציבורי/פרטי) ושולח את המפתח הציבורי בחזרה לשרת הדירקטור.
4. **עדכון רשימת המפתחות בשרת הדירקטור**: שרת הדירקטור מעדכן את רשימת המפתחות המנוהלת בבסיס הנתונים שלו. הרשימה כוללת את המפתחות הפומביים של כל הצמתים הפעילים ברשת, ובאמצעותה כל צומת יכול לבצע הצפנה עם הצמתים האחרים.
5. **אישור חיבור והתחלת שימוש ברשת**: לאחר קבלת המפתח הציבורי מהצומת ועדכון בסיס הנתונים, שרת הדירקטור שולח אישור חיבור לצומת, שמאפשר לו להשתתף ברשת בצורה מלאה ולשלוח חבילות מוצפנות לצמתים אחרים.

—-----------------------------------------------------

**הצומת האחרון במסלול -**   
  
**קבלת החבילה**: הצומת האחרון מקבל את החבילה לאחר שעברה דרך כל הצמתים הקודמים. כל אחד מהצמתים הקודמים פענח שכבה אחת של הצפנה, ולכן החבילה המגיעה לצומת האחרון עדיין מוצפנת עם המפתח הציבורי שלו.

**פענוח השכבה האחרונה**: הצומת האחרון משתמש במפתח הפרטי שלו כדי לפענח את השכבה האחרונה של ההצפנה. זהו השלב שבו כל שכבות ההצפנה שהתווספו לאורך הדרך מפוענחות, והמידע המקורי שהמשתמש שלח מתגלה.

**עיבוד המידע**: לאחר הפענוח, הצומת האחרון יכול לעבד את המידע המתקבל בהתאם לדרישות הרשת או המשימה שבוצעה. במקרים מסוימים, הצומת האחרון הוא גם היעד הסופי של המידע, ולכן הוא עשוי להיות אחראי על פעולה נוספת כמו שליחת תשובה חזרה.

**שבירת אנונימיות מוגבלת**: הצומת האחרון יודע שהוא היעד הסופי של החבילה, ולכן הוא יודע מאיזה צומת קיבל את המידע האחרון. עם זאת, הוא אינו יכול לדעת את המסלול המדויק שהחבילה עברה או את זהות המשתמש המקורי ששלח אותה, שכן כל הצמתים הקודמים לא חשפו את זהות השולח אלא רק את הצומת הבא

**קלט -** החבילה המוצפנת עם המידע

**פלט -** המידע המפוענח שמועבר ליעד  
  
**הסבר קוד**

במערכת Onion Routing, התפקיד של הצומת האחרון הוא לפענח את השכבה האחרונה של ההצפנה ולהעביר את המידע ליעד הנכון. הצומת האחרון מבצע את הפעולות הבאות:

1. **קבלת החבילה**:  
   הצומת האחרון מקבל את החבילה לאחר שהיא עברה דרך מספר צמתים, כאשר כל צומת קודם פענח שכבה אחת של ההצפנה. החבילה שהגיעה לצומת האחרון עדיין מוצפנת בשכבה האחרונה עם המפתח הציבורי של הצומת הזה.
2. **פענוח השכבה האחרונה**:  
   הצומת האחרון משתמש במפתח הפרטי שלו כדי לפענח את השכבה האחרונה של ההצפנה. אחרי פענוח זה, מתקבל המידע המקורי שנשלח על ידי המשתמש.
3. **עיבוד המידע**:  
   לאחר פענוח החבילה, הצומת האחרון מעבד את המידע. אם הצומת הזה הוא היעד הסופי, הוא עשוי לשלוח תשובה חזרה או לבצע פעולה נוספת בהתאם לדרישות.
4. **שמירה על אנונימיות מוגבלת**:  
   הצומת האחרון יודע רק מי הצומת האחרון שקיבל ממנו את המידע, אך אינו יודע את המסלול המלא או את זהות השולח המקורי, מה ששומר על רמת אנונימיות חלקית.

—-----------------------------------------------------

**בחירת המסלול דרכו תעבור החבילה -**

המערכת תחשב מסלול באופן רנדומלי

הבחירה הרנדומלית של המסלול שבו תעבור החבילה בתהליך ה-Onion Routing נועדה להבטיח אנונימיות וקשיי מעקב עבור כל חבילה שנשלחת דרך הרשת. השימוש באלגוריתם רנדומלי מאפשר למערכת לבחור צמתים בצורה בלתי צפויה, כך שלא יהיה קל לגלות את המקור או היעד של החבילה.

### למה משתמשים בבחירה רנדומלית:

1. **שמירה על אנונימיות**:  
   מסלול רנדומלי מקשה על כל גורם חיצוני לדעת את המסלול המדויק שהחבילה עוברת בו, ובכך משמר את פרטיות השולח והמקבל.
2. **מניעת מעקב תעבורה**:  
   בחירה בלתי צפויה של צמתים מקשה על גופי מעקב לנסות לשחזר את הנתיב שדרכו עברה החבילה, ולמנוע איתור של זהות המשתמשים.
3. **מניעת התקפות על צמתים**:  
   אם התוקפים לא יודעים אילו צמתים ישתמשו בהם בתקשורת, הם לא יכולים לתקוף או לשבש את המסלול באופן יעיל.

—-----------------------------------------------------

**העברת המידע דרך הצמתים -**

[הפקטה](#7n3q9x4ctviu) עוברת בין מספר צמתים במטרה להגן על המידע ולשמור על פרטיות המשתמש. ההודעה מוצפנת בשכבות, מספר השכבות כמספר הצמתים במסלול. כל צומת מפענחת את השכבה הקשורה אליה וחושפת את המידע של הצומת הבאה אליה היא צריכה להעביר את הפקטה בלבד, המידע נשאר מוצפן. הצומת לא יודעת מי המקור ומי היעד אלא רק מי הצומת ממנה קיבלה את הפקטה ואל מי היא צריכה להעבירה. כך אף אחד לא יכול לעקוב אחר החבילה ולמצוא את מקורה או מטרתה.

**צומת (Node)**

*מרכיב פרטי -*

המרכיב הפרטי של הצומת אחראי על שליחת פקטות חדשות מהלקוח השולח:

**1**) הלקוח מבקש את כל [המפתחות של הצמתים](#o4hygwrl36sr) ברשת ומחשב את המסלול של הפקטה.

**2**) לאחר קביעת המסלול, הלקוח מצפין את הפקטה בהצפנת AES בהתאם למספר הצמתים ולמפתחות של כל אחד מהם, ויוצר שכבות הצפנה לפי המבנה שנקבע (headers, data).

**3)** הפקטה נשלחת לצומת הראשון במסלול.

*מרכיב חיצוני -*

המרכיב החיצוני של הצומת אחראי על העברת המידע דרכו, מבלי להיות קשור אליו ישירות:

**1**) הצומת מקבל חבילה מוצפנת באמצעות הצפנה א-סימטרית עם המפתח שנוצר ונשלח לשרת הדירקטורי.

**2**) תפקידו של הצומת הוא לפענח את החבילה ולהוציא מתוכה את החבילה הבאה ואת הכתובת אליה יש לשלוח אותה.

**3**) הצומת שולח את החבילה המוצפנת, שניתנת לפענוח רק על ידי הצומת הבא במסלול

**פקטה של רשת טור -**

מכילה את ההודעה של הלקוח (המידע המקורי) ומספר שכבות הצפנה "הנמצאות מסביב למידע" ומצפינות אותו. הפקטה נשלחת מהמקור כאשר היא מכילה את כל שכבות ההצפנה הרלוונטיות לנתיב שתעבור בדרך עד ליעדה (מספר שכבות ההצפנה כמספר הצמתים שתעבור בדרך). שכבת ההצפנה החיצונית ביותר תוסר על ידי הצומת הראשונה וכך הלאה, לאחר ששכבה מוסרת השכבה הבאה נחשפת יחד עם פרטי הניתוב הרלוונטיים לצומת הבאה בלבד ואיננה מודעת לשאר החוליות בנתיב.

רשת טור משתמשת בתאים (יחידות נתונים פנימיות שלה) בגודל קבוע, לרוב 512 בתים, בשביל לטפל בנתונים בתוך הרשת.

כל תא "מעביר" מכיל :

קובץ header המכיל פרטי ניתוב ובקרה (64 בתים) -   
 IPdestination - 4 byts  
 PORTdestination - 4 byts

אורך המידע - 2 בתים

פקודה (סוג התא - העברה, יצירה, השמדה)- בית אחד

ID של המסלול - 2 בתים

ID של זרם הנתונים הספציפי מהמסלול - 2 בתים

* השדות הקבועים כוללים מידע בקרה לניתוב ועיבוד התא.

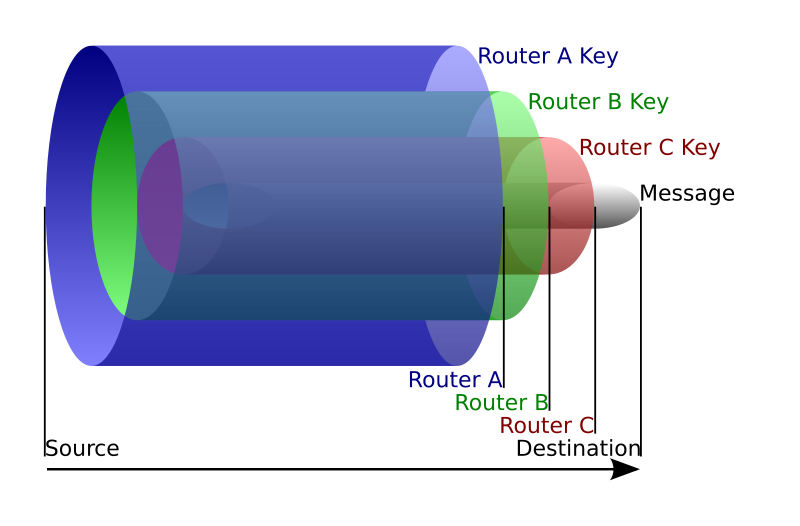
ומטען של המידע המוצפן - (448 בתים)

הפקטה מחולקת למספר תאים בגודל קבוע, כל תא מוצפן מספר פעמים (ככמות הצמתים). התאים מועברים כמטען של הפקטות וכל צומת מפענחת שכבה אחת וחושפת את השכבה החדשה ופרטי המעבר אליה. החוליה האחרונה מרכיבה מחדש את הפקטה המקורית מהתאים המפוענחים ואותה פקטה נשלחת אל היעד.

יצירת פקטה -

* בחירת סוג התא ("לחיצת יד" - תחילת התקשורת בין צמתים, "העברה" - מעבר מידע בין צמתים, "פקודה" - פקודות בקרה), הכנת המידע במטען ויצירת קובץ header
* הרכבת התא והצפנתו
* שליחת התא לצומת במסלול

**דוגמה:**



**קלט -** הפקטה שצריכה לעבור במסלול בצורה מאובטחת

**פלט -** המידע הגיע ליעד האחרון לאחר שעבר במסלול

### הסבר קוד

1. הלקוח יוצר מסלול רנדומלי דרך Client::SelectRoute().
2. הלקוח מצפין את הפקטה בשכבות באמצעות Client::EncryptPacket(), כשהשכבות מוצפנות לפי מפתחות הצמתים שבחר.
3. הפקטה נשלחת לצומת הראשון בעזרת Client::SendPacket().
4. הצומת הראשון מקבל את הפקטה דרך Node::ReceivePacket(), מפענח שכבה אחת עם Node::DecryptLayer(), ומעביר את הפקטה הלאה עם Node::ForwardPacket().
5. כך התהליך נמשך עד שהפקטה מגיעה ליעד, שם היא מפוענחת במלואה.

—-----------------------------------------------------

**ניהול הצמתים -**

ניהול הצמתים ברשת Onion Routing מתבצע על ידי **שרת הדירקטורי**, שהוא רכיב מרכזי שמבטיח את יציבות הרשת, האבטחה שלה, ואת תפקוד הצמתים. תפקידו של השרת הוא לנהל מספר היבטים קריטיים כמו הצפנות, ניטור הצמתים, ואיסוף והפצת המפתחות הפומביים בצורה יעילה.

### תפקידים מרכזיים של שרת הדירקטורי:

#### 1. ניהול מפתחות והצפנות:

השרת הדירקטורי אחראי על ניהול כל תהליך ההצפנה ברשת, שמתחיל ביצירת ואיסוף המפתחות הפומביים של הצמתים.

* **בזמן התחברות צומת חדש**: השרת מבקש מהצומת ליצור זוג מפתחות הצפנה (מפתח פרטי ומפתח פומבי). המפתח הפומבי נשלח לשרת והוא מעדכן את כל הצמתים ברשת במפתחות החדשים.
* **הפצת המפתחות**: השרת יוצר מסמכים המתארים את המפתחות הפומביים של כל הצמתים ברשת. המסמכים הללו מופצים ללקוחות, שמורידים אותם ובונים על פיהם את מסלול החבילה דרך הצמתים השונים.

#### 2. ניטור הצמתים:

שרת הדירקטור מנטר באופן רציף את הפעילות של הצמתים כדי לוודא שכולם פועלים כראוי.

* [**בדיקות תקינות**](#lhj6y6ldjiot): השרת שולח הודעות לצמתים כדי לוודא שהעברת המידע מתבצעת בצורה תקינה. אם צומת מסוים לא מגיב או חווה תקלות, השרת מעדכן את כל הרשת ומוציא את הצומת מהמסלול האפשרי של החבילות.
* **סטטוס צמתים**: השרת שומר על סטטוס עדכני של כל צומת ברשת, כולל האם הצומת פעיל או "חולה" (לא מגיב או נתקל בבעיות טכניות).
* **זיהוי דפוסים חריגים**: השרת משתמש בטכניקות ניטור לזיהוי דפוסים חריגים, כמו תעבורה מוגברת   
  או פעולות חריגות מצומת מסוים, כדי למנוע פעילות זדונית.

**הסבר הקוד**

במערכת ה-Onion Routing, ניהול הצמתים מתבצע באמצעות **שרת הדירקטורי**, אשר אחראי על תפעול תקין של הרשת, כולל ניהול הצפנות, ניטור צמתים, והפצת מפתחות פומביים לצמתים. השרת משתמש במספר פונקציות עיקריות כדי לנהל את הצמתים ולוודא את יציבות ואבטחת הרשת.

1. **ניהול מפתחות והצפנות**:
   * הפונקציה CollectPublicKeys() אחראית על איסוף המפתחות הפומביים מכל הצמתים החדשים שהצטרפו לרשת.
   * DistributePublicKeys() אחראית על הפצת המפתחות הפומביים לצמתים ולקוחות הרשת. פונקציה זו מאפשרת ללקוחות לבנות את המסלול על סמך המפתחות.
   * כאשר צומת חדש מצטרף לרשת, הפונקציה RequestKeyPair() מופעלת כדי לדרוש מהצומת לייצר זוג מפתחות (מפתח פרטי ומפתח פומבי).
2. **ניטור הצמתים**:
   * הפונקציה MonitorNodeHealth() מנטרת את תקינות הצמתים. היא שולחת הודעות בדיקה כדי לוודא שהצומת פעיל ומעביר חבילות בצורה תקינה.
   * אם צומת לא מגיב, הפונקציה MarkNodeAsInactive() מסמנת אותו כלא פעיל, ומעדכנת את כל הרשת דרך UpdateNetworkStatus().
   * פונקציית DetectAnomalies() מזהה דפוסים חריגים בתעבורת הצמתים, כמו תעבורה מוגברת או פעולות חשודות, כדי למנוע מתקפות או פעילות זדונית.

—-----------------------------------------------------

**מימוש פרוטוקול TCP -**

**קלט:** החבילה להעברה

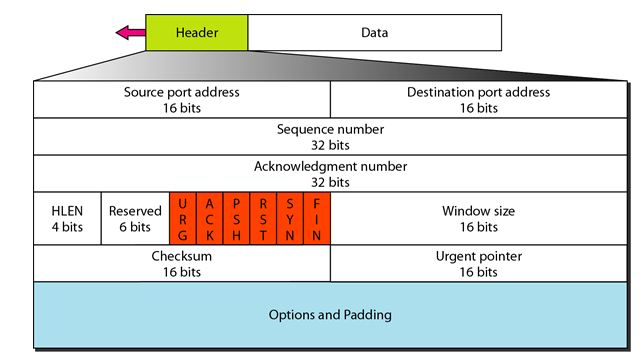
**פלט:** סטטוס בהתאם לריצה והצלחתה

**מה הוא עושה:** פרוטוקול TCP (פרוטוקול בקרת שידור) הוא פרוטוקול ליבה בשכבת התעבורה, המבטיח מסירה אמינה, מסודרת ונבדקת של שגיאות נתונים בין יישומים הפועלים על מארחים המתקשרים באמצעות רשת IP בין 2 תחנות ברשת.

**איך הוא עושה את זה:**

תכונות עיקריות של TCP כוללות:

**חלוקת המידע לחלקים:** **הודעת segment) TCP)**



הודעת TCP נקראת סגמנט. הסגמנט מורכב משני חלקים עיקריים, header וdata.

**data** הוא בעצם שכבת האפליקציה (החלק הספציפי בבאפר שמעבירים - אנקפסולציה).

האנקפסולציה = כל שכבת האפליקציה הופכת בעצם להיות ה-DATA של שכבת התעבורה. כלומר החבילה של שכבת התעבורה כוללת הדרים של TCP ואז את כל שכבת האפליקציה בתור DATA.

ל-TCP אין data משלו, הוא רק מוסיף את ההדרים לdata הקיים.

**header** מורכב גם הוא מחלקים שונים המכילים את כל המידע הרלוונטי:

* פורט יעד ופורט מקור, בדומה ל-UDP, והריבוב מתבצע בשניהם באותו האופן.
* ה seq number ו- ack number
* HLEN כולל את אורך כל החלק של ההדרים ב- TCP. אורך זה יכול להשתנות מכיוון שיש אפשרות להוסיף אופציות נוספות. לרוב אפוציות אלא לא בשימוש. נמדד בכפולות של 4 בתים (32 ביטים).
* RESERVED לא בשימוש - 3 אפסים
* Checksum הוא עיבוד של כל המידע (הדר + דאטה. ההדר הוא הדר מיוחד שנוצר עבור הצ'קסאם) עבור בדיקה של אמינות. ב-TCP בניגוד ל-UDP קיימת משמעות חשובה ל-Checksum. הוא נבדק בכל פעם וחבילה שלא עברה את מבחן הצ'קסאם **לא תאושר ע"י ACK.**
* Urgent Pointer – מסמן את המקום "הדחוף" במידע, בא ביחד עם הדגל של URG.
* Options and padding – אפשרות להרחבת הפרוטוקול לאופציות נוספות. לא בשימוש לרוב.

שני חלקים חשובים בהדרים הינם:

**הדגלים :**

 לכל אחד מהם מוקצה רק ביט אחד, ולכן - **דלוק (1)** או **מכובה (0).**

**ACK =** מסמן שההודעה כוללת מספר Ack כלומר היא מאשרת קבלה של הודעה קודמת.  
 משמש גם בלחיצת יד משולשת.

**SYN =** משמש בלחיצת היד המשולשת.

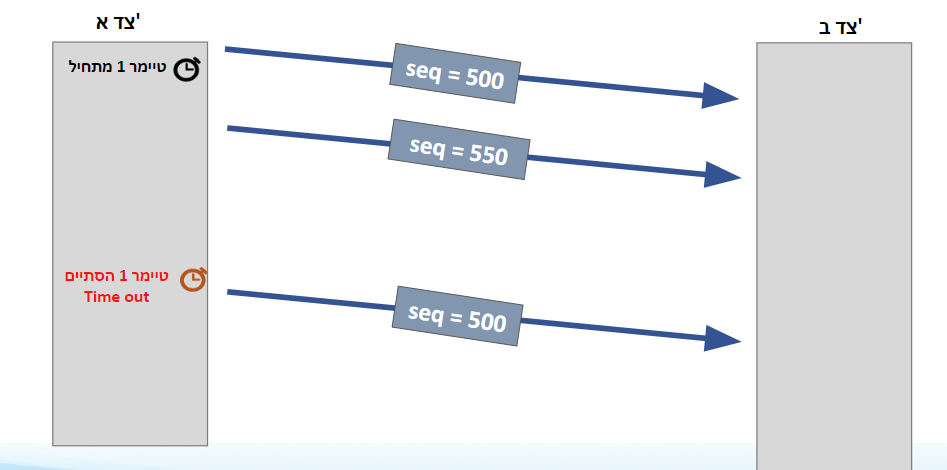
**FIN =** משמש בהליך סיום התקשורת.

* **PSH** - נעשה שימוש בו כאשר רוצים שהחבילה תשלח באופן מיידי ולא בחלקים בגדלים שהTCP היה רוצה.
* **URG -** הדגל מאפשר לסמן חלק מסוים במידע שהוא דחוף יותר ויועבר קודם לעיבוד. לשם כך צריך להדליק את הדגל וגם לשים בשדה URG POINTER את מיקום המידע הדחוף.
* **NS, ECE, CWR -** דגלים אלו קשורים לבקרת גודש (ECN). זוהי הרחבה ל-TCP/IP שהוצגה בשנת 2001 והיא אופציונלית. היא מאפשרת למחשבים מתקשרים ליידע אחד את השני על עומס ברשת כדי שיוכלו להאט את קצב ההעברה בהתאם, ובעצם מחליפה את המנגנון הרגיל שבו רק במידה וחבילות אבדו יואט הקצב.

**גודל חלון:**

גודל חלון הוא שדה שנועד לאפשר תכונה נוספת של TCP שנקראת Flow Control (שליטה בקצבים).

כדי להימנע ממצב בו צד אחד מעמיס מידע בקצב מהיר מדי עבור הצד השני, כל צד שולח לצד השני את גודל החלון שלו (לכל צד יש ערך אחר למספר הזה).  
  
**יכולים להיות גודלי חלון שונים לכל צד-** הדברתלוי בהחלטת הצדדים.



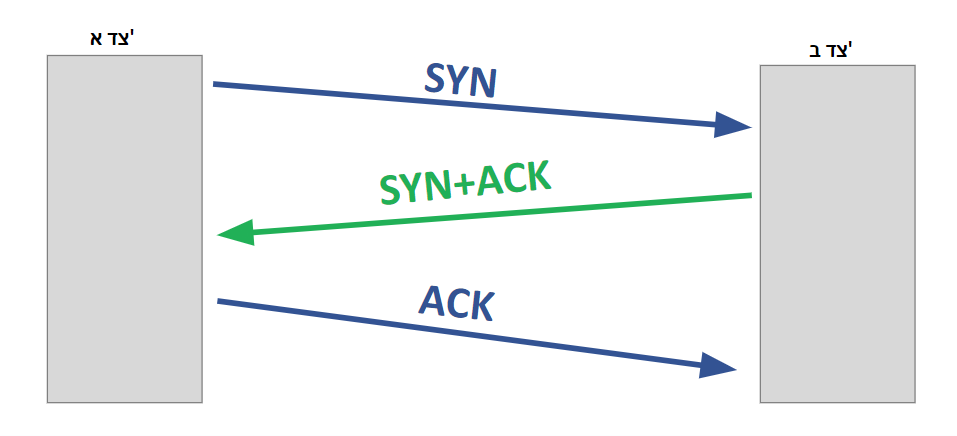
**יצירת חיבור: TCP משתמש בלחיצת יד תלת כיוונית כדי ליצור חיבור בין הלקוח לשרת.**

לחיצת יד משולשת היא הדרך של שני הצדדים להתחיל שיחה על ידי פרוטוקול TCP  
 ובגלל שהם עושים זאת הם מחליפים את הנתונים הנדרשים להמשך השיחה.   
 ללחיצת היד המשולשת יש שתי מטרות עיקריות:

* + לוודא שהפורט בצד השני פתוח ומאזין לחיבור.
  + לסנכרן את שני הצדדים על מספרי ה-Seq ההתחלתיים (ISN) שכל צד בחר.

**SEQ** = המספר של הבית הראשון באותה החבילה  
 לדוגמא המספר של החבילה הוא 10. הוחלט לחלק את החבילות ל 16 ביטים כל אחת   
 ואז בחבילה ה 10 המספר SEQ יהיה 145 בהנחה שהתחילו ב 1  
   
 **LEN** = המספר המייצג את גודל החבילה   
 במקרה של יצירת החיבור, ה LEN יהיה 0 כי הוא לא שולח data

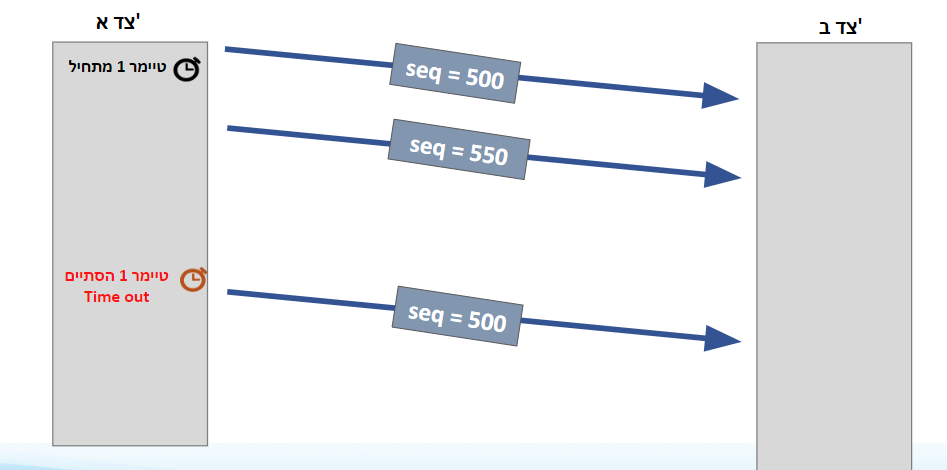
בלחיצת היד המשולשת הצדדים מפעילים דגלים מסוימים   
 בהודעה הראשונה הצד היוזם שולח הודעה עם הדגל SYN דלוק  
 בשביל להראות לצד השני שהוא מעוניין להתחיל שיחה על ידי פרוטוקול TCP  
 ההודעה השניה הנשלחת על ידי הצד השני   
 שולח הודעה עם שני דגלים SYN ו ACK   
 SYN מראה שהוא מעוניין להתחיל שיחה   
 ו ACK מראה שהוא קיבל את החבילה הקודמת



**טיימר:**

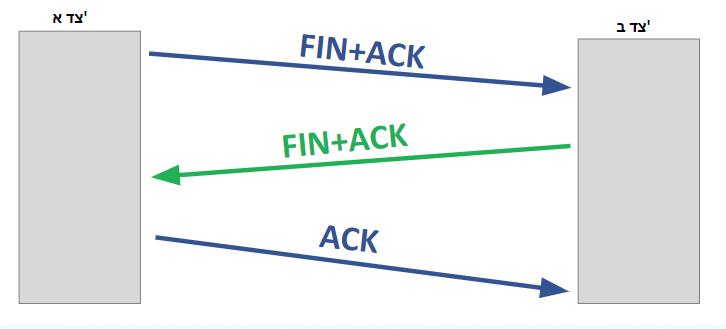
הטיימר ב-TCP ממלא תפקיד קריטי בהבטחת אמינות ההעברה, שהיא אחת מהמטרות המרכזיות של הפרוטוקול. מכיוון ש-TCP נועד לספק העברת נתונים בטוחה וללא אובדן, הטיימר נועד להבטיח שכל חבילה שנשלחה תגיע ליעדה.

כאשר חבילה נשלחת, המערכת מפעילה טיימר שמתחיל לספור לאחור לזמן מסוים, אשר מבוסס על הערכות זמן ההגעה המשוערות. אם הזמן חולף מבלי שהתקבלה תשובה או אישור (ACK) מהצד המקבל, הדבר מעיד על כך שהחבילה לא הגיעה ליעד, והטיימר גורם לשליחה מחדש של החבילה. בצורה זו, TCP מתמודד עם בעיות כמו אובדן נתונים או שגיאות בקו התקשורת, תוך שמירה על העברת נתונים אמינה.



**סגירת חיבור: TCP משתמש בלחיצת יד תלת כיוונית כדי לסגור חיבור בין הלקוח לשרת.**כמו ביצירת חיבור יש שלושה שלבים לסגירת חיבור בהודעה הראשונה הצד שמעוניין לסגור את החיבור שולח לצד השני הודעה בה הוא מדליק שתי דגלים ACK ו FIN  
ACK בשביל שידע שקיבל את ההודעה הקודמת ו - FIN בשביל שידע שהוא מעוניין לסגור את החיבור   
   
בשלב השני הצד השני שולח הודעה בה הוא מדליק שתי דגלים ACK ו FIN  
ACK בשביל שידע שקיבל את ההודעה הקודמת ו - FIN בשביל שידע שהוא מעוניין לסגור את החיבור

בשלב השלישי הוא מדליק רק דגל אחד ACK בשביל שידע שקיבל את ההודעה הקודמת   
ובכך הצדדים מסיימים לשוחח



**בעיות ומחירי השימוש ב-TCP -**

**המידע הנוסף ש-TCP מעביר לעומת UDP -**

הTCP מכיל מידע רב נוסף מה שמוביל לזמן שימוש רב יותר, דבר קריטי בעולם האינטרנט של ימינו.

* הדר גדול יותר לכל חבילה (20 בתים לעומת 8)
* לחיצת יד (שמוסיפה עוד 3 חבילות ~ 150-200 בתים)

**תקורה (Overhead)** -

בתקשורת, תקורה היא כל המידע שאני מעביר שהוא לא ההודעה עצמה שרציתי להעביר. בשכבות – המידע הוא שכבת האפליקציה. כל שאר השכבות הן התקורה.

**אבטחה ומתקפות -**

**מתקפת dos ו ddos:**

מתקפות (Denial of Service) ו- (Distributed Denial of Service) הן סוגי מתקפות שמכוונות לשבש או להשבית שירותים מקוונים על ידי העמסת יתר על משאבי השרתים או הרשתות. בפרוטוקול TCP, אחת השיטות הנפוצות למתקפת DoS היא "SYN Flood". בפרוטוקול TCP, כאשר יוזמים חיבור, התהליך מתחיל עם שלב שנקרא "לחיצת יד משולשת" (Three-way handshake). במתקפת SYN Flood, התוקף שולח מספר רב של בקשות SYN לשרת, אך אינו משלים את תהליך החיבור, מה שגורם לשרת להחזיק משאבים עבור חיבורים שלא הושלמו. זה מוביל להתשה של המשאבים הזמינים בשרת, עד שהוא אינו יכול לקבל בקשות נוספות ממשתמשים לגיטימיים.

במתקפת DDoS, המתקפה מבוצעת על ידי מספר גדול של מכשירים, לרוב מחשבים שנשלטים על ידי תוכנות זדוניות (בוטים), שמבצעים את אותה פעולת עומס בו-זמנית ממיקומים שונים. כתוצאה מכך, השרת או הרשת מוצפים בכמות עצומה של תעבורה מזויפת שמקשה עוד יותר על זיהוי וסינון המתקפה. בעוד שמתקפת DoS מגיעה ממקור אחד, מתקפת DDoS הרבה יותר קשה להדיפה, כיוון שהיא מפוזרת גיאוגרפית ומגיעה ממספר רב של מכשירים.

SYN Flood הוא סוג של מתקפת dos שמתרחשת בלי הצורך להתחבר לשרת זה  
 מתבצע על ידי שליחה של בקשת התחלת שיחה ובכך מציף את השרת ביותר מדי  
 בקשות וגרם לשרת לקרוס ולא לענות לבקשות אמיתיות

**סיכום:**  
**העברת נתונים:** TCP מבטיח מסירה אמינה של נתונים באמצעות מנגנונים כמו מספרי sequence acknowledgments ו [retransmissions](#5h1ud5jt9k4o).

**בקרת זרימה:** TCP משתמש במנגנון חלון הזזה כדי לשלוט בזרימת הנתונים בין השולח למקלט.

**בקרת גודש:** TCP משתמש באלגוריתמים כמו התחלה איטית, הימנעות מגודש, שידור חוזר מהיר והתאוששות מהירה כדי לנהל עומס ברשת.

**סיום חיבור:** TCP משתמש בתהליך בן ארבעה שלבים לסגירת חיבור "gracefully" .

[**מבנה Packet**](#y7vsm6njww7v) **:** הגדרת מבנה של "packets TCP", כולל "header" ו "payload" . צריך לטפל באופן ידני בשדות כמו יציאות מקור ויעד, מספרי "sequence" ו "acknowledgment" , דגלים (SYN, ACK, FIN וכו'), גודל חלון, "checksum" ו "urgent pointer" .

[**לחיצת יד תלת כיוונית**](#i8l5osx54exn) **- תהליך יצירת הקשר:**

שליחת חבילת SYN לשרת.

קבלת חבילת SYN-ACK מהשרת.

שליחת חבילת ACK לשרת כדי ליצור את החיבור.

העברת נתונים ואמינות - הטמעת מנגנונים להעברת נתונים אמינה:

שמירה על רצף ומספרי אישור.

הטמעת [חלון הזזה](#u68iunml4bmp) לבקרת זרימה.

טיפול בשידורים חוזרים עבור מנות שאבדו.

בקרת זרימה: יישום פרוטוקול חלון ההזזה כדי להבטיח שהשולח לא יציף את המקלט בכמות גדולה מדי של נתונים בבת אחת.

בקרת גודש: הטמעת אלגוריתמים בסיסיים של בקרת גודש כדי למנוע עומס ברשת.

[סיום חיבור](#csohm3z94f4j): יישום תהליך בן ארבעת השלבים לסגירת חיבור "gracefully" :

שליחת חבילת FIN לשרת.

קבלת ACK עבור חבילת FIN.

קבלת חבילת FIN מהשרת.

שליחת ACK עבור חבילת FIN של השרת.

### מבנה הקוד:

**ספריות כלולות**:

* + <iostream>: לטיפול בקלט ופלט.
  + <cstring>: לטיפול במחרוזות.
  + <sys/types.h>, <sys/socket.h>, <netinet/in.h>, <arpa/inet.h>, <unistd.h>: לטיפול בסוקטים ופעולות רשת.

-----------------------------------------------------

### העברת קבצים ברשת TOR -

TOR מבטיחה שהעברת הקבצים תתבצע באופן פרטי, תוך שימוש בהצפנה מרובת שכבות (Onion Routing) שמונעת מעקב אחר נתוני השולח והמקבל.

#### שלבי העברת הקובץ ברשת TOR:

1. **חלוקת הקובץ לסגמנטים:** העברת קבצי MP4 גדולים נעשית באמצעות חלוקתם למקטעים קטנים, הנקראים סגמנטים. תהליך זה מתבצע בפרוטוקול TCP, בו הקובץ מחולק למקטעים בגודל המותאם לרשת (לרוב 1460 בתים בכל סגמנט) להעברה בטוחה ואנונימית דרך צמתים שונים ברשת.
2. **הצפנה מרובת שכבות (Onion Encryption):** כל סגמנט מוצפן בכמה שכבות על פי הצמתים שדרכם הוא יעבור. כל צומת מפענח את השכבה המתאימה לו ומעביר את הסגמנט לצומת הבא. תהליך זה מבטיח שהמקור והיעד של הסגמנט נותרים אנונימיים.
3. **העברת הסגמנטים בצמתים:** הסגמנטים עוברים דרך מספר צמתים ברשת TOR, כשכל צומת יודע רק מאיפה הוא קיבל את המידע ולאן עליו להעבירו. בכך, נשמרת פרטיות המשתמשים ברשת והמידע מוגן ממעקב.
4. **הרכבת הקובץ בצד היעד:** כשהסגמנטים מגיעים ליעדם, השרת או הלקוח בונים מחדש את הקובץ על פי מספרי הרצף שהגיעו בכותרות TCP. כך נוצר קובץ MP4 שלם שניתן להורדה או צפייה בצד הלקוח.

#### יתרונות:

* **אבטחה ואנונימיות**: תהליך הצפנת השכבות מבטיח שהמעבר בין הצמתים נעשה באופן אנונימי ובטוח.
* **חלוקת עומס**: חלוקת הקובץ למקטעים מאפשרת העברה יעילה יותר, עם הפחתת הסיכון לנפילת חבילות ברשת.

**חסרונות:**

* **זמנים:** העברה איטית יותר בעקבות השימוש במנגנונים השונים.

—------------------------------------------------------------

**הצפנה אסימטרית עם מפתחות סימטריים -**

הצפנה א-סימטרית עם מפתחות סימטריים ברשת Tor היא אחת מהשיטות המרכזיות להבטחת פרטיות ואנונימיות של המשתמשים תוך כדי מעבר המידע בין מספר צמתים ברשת.

### מה זה עושה?

### המערכת משתמשת בהצפנה א-סימטרית (RSA) על מנת להעביר מפתחות סימטריים בין הצמתים השונים (Entry, Middle, Exit). לאחר החלפת המפתחות בצורה מאובטחת, נעשה שימוש בהצפנה סימטרית (AES) להעברת ההודעות עצמן בין הצמתים בצורה מוצפנת ויעילה.

### הצפנה סימטרית:

הצפנה סימטרית היא שיטת הצפנה שבה אותו מפתח משמש גם להצפין את המידע וגם לפענח אותו. כלומר, לשני הצדדים המשתתפים בתקשורת יש את אותו מפתח סודי, והם משתמשים בו כדי לבצע את ההצפנה והפענוח של הנתונים. [פרטי המפתחות הפומביים](#_by3dtxdxmuhf) מאוחסנים בקובץ בשרת דיירקטורי.

### הצפנה א-סימטרית:

### בהצפנה א-סימטרית יש שני מפתחות שונים: מפתח ציבורי ומפתח פרטי. המפתח הציבורי משמש להצפין את המידע, והמפתח הפרטי, שנשאר סודי אצל המקבל, משמש לפענח את המידע. כלומר, לכל אחד יש מפתח ציבורי שניתן לשתף עם אחרים, ומפתח פרטי שנשאר אצלו בלבד.

### קלט:

* **מפתחות ציבוריים ופרטיים** של כל אחד מהצמתים (הצפנה א-סימטרית).
* **נתונים או הודעות** שמועברים ברשת Tor.

### פלט:

* **מפתח סימטרי מוצפן** המשותף בין כל צומת (בשלב ההחלפה).
* **הודעה מוצפנת בשכבות (Onion Encryption)** שהולכת דרך כל הצמתים עד שהיא מגיעה ליעד הסופי.

### איך זה עובד?

1. **החלפת מפתחות א-סימטרית**: כל צומת ברשת Tor (כניסה, אמצע, יציאה) מייצר זוג מפתחות – ציבורי ופרטי. הצומת הראשון (Entry) מצפין את המפתח הסימטרי של כל שכבה במפתח הציבורי של כל צומת בנתיב.
2. **הצפנה שכבתית (Onion Encryption)**: כל שכבה בהודעה מוצפנת בנפרד באמצעות מפתח סימטרי שנבחר בצורה מאובטחת. כשכל צומת מקבל את ההודעה, הוא מפענח רק את השכבה הרלוונטית לו, מעביר את ההודעה המפוענחת לצומת הבא, ושומר על האנונימיות של השולח והנתיב.
3. **פענוח הסופי**: כשההודעה מגיעה לצומת היציאה (Exit), שכבת ההצפנה האחרונה מפוענחת וההודעה נשלחת ליעד הסופי (אינטרנט רגיל). אף אחד מהצמתים בדרך לא יודע את המקור או היעד של ההודעה.

### יתרונות:

* השימוש בהצפנה א-סימטרית מבטיח שהחלפת המפתחות בין הצמתים תהיה מאובטחת.
* לאחר החלפת המפתחות, השימוש במפתחות סימטריים מאפשר הצפנה מהירה ויעילה של ההודעות, תוך שמירה על אנונימיות ופרטיות המשתמש.

—------------------------------------------------------------

### שמירת מפתחות פומביים של הצמתים -

ניהול המפתחות ברשת מתבצע על ידי שרת הדירקטורי, והוא אחראי לאחסן את המפתחות הפומביים של הצמתים בצורה מאובטחת ויעילה. אחת הדרכים לניהול המפתחות היא על ידי שימוש במסד נתונים כמו **SQLite3**, שבו נשמרים כל המפתחות הפומביים של הצמתים בטבלה ייעודית. המידע על הצמתים נשמר במסד נתונים בצורה מובנית, כך ששרת הדירקטורי יוכל לשלוף ולעדכן את המפתחות בקלות בהתאם לצורך.

#### מבנה הטבלה:

בטבלת הצמתים נשמרים מספר שדות קריטיים:

1. **PORT**: שדה מסוג **Integer**, המייצג את הפורט שבו פועל הצומת.
2. **IP**: שדה מסוג **Integer**, המייצג את כתובת ה-IP של הצומת.
3. **PUBLIC\_KEY**: שדה מסוג **Integer**, שבו נשמר המפתח הפומבי של הצומת.

בעת הצטרפות של צומת חדש לרשת, שרת הדירקטורי מבקש מהצומת ליצור זוג מפתחות: מפתח פרטי ומפתח פומבי. המפתח הפומבי נשלח לשרת ונשמר בטבלת הצמתים במסד הנתונים.

#### 

#### תהליך השמירה:

1. **יצירת מפתחות**: בזמן התחברות של צומת חדש, הצומת מייצר זוג מפתחות (מפתח פרטי ומפתח פומבי).
2. **שליחת המפתח הפומבי**: המפתח הפומבי נשלח לשרת הדירקטורי, שם הוא נשמר בטבלת הצמתים.
3. **אחסון המפתח**: שרת הדירקטורי מאחסן את המפתח הפומבי יחד עם פרטי הצומת (IP ו-Port) בטבלה ייעודית במסד הנתונים.

#### יתרונות השימוש ב-SQLite3:

* **ביצועים גבוהים**: מאגר קל וקומפקטי שמאפשר שליפה מהירה ועדכונים יעילים.
* **ניידות**: SQLite3 לא דורש שרת מסד נתונים חיצוני, ולכן הוא מתאים לרשתות מבוזרות או קטנות.
* **אבטחה**: נתונים שמורים בצורה מסודרת וניתנים לניהול גישה לפי פרוטוקולי אבטחה נדרשים.

—------------------------------------------------------------

### בדיקת תקינות צמתים -

ניהול תקינות הצמתים ברשת מבוצע על ידי שרת הדירקטורי, שתפקידו לוודא שכל הצמתים פועלים בצורה תקינה וממשיכים להשתתף בהעברת החבילות ברשת בצורה אמינה ובטוחה. אחת השיטות היעילות לבדיקת הצמתים היא שליחת בקשות אקראיות לצמתים כדי לבחון את זמינותם ואת יכולתם לטפל בחבילות המידע.

#### תהליך בדיקת הצמתים:

**שליחת בקשת בדיקה**: השרת הדירקטורי שולח לצמתים בקשות בדיקה בפרקי זמן קבועים או אקראיים, שבהן הצומת נדרש להגיב תוך זמן מוגדר מראש. בקשת הבדיקה נועדה לוודא שהצומת זמין ויכול לטפל בחבילות.  
  
 **1. מבנה הבקשה**

בקשת בדיקה שנשלחת מהשרת הדירקטורי לצומת ברשת Onion Routing עשויה לכלול את המרכיבים הבאים:

* **מזהה הבקשה (Request ID)**: מזהה ייחודי שנועד לאפשר לשרת לעקוב אחרי הבקשה ולזהות אותה בתגובה.
* **תאריך ושעה**: הזמן שבו נשלחה הבקשה, מה שמסייע בניתוח נתונים ובביצוע הערכות מאוחרות.
* **מפתח ציבורי של השרת**: המפתח הציבורי של השרת, המאפשר לצמתים לאמת את מקור הבקשה.
* **שדה לאימות**: שדה זה כולל Hash שנוצר בעזרת אלגוריתם הצפנה (כמו SHA-256) על מנת להבטיח שהבקשה לא שונתה במהלך ההעברה.

**מבנה הבקשה יכול להיראות כך (בפשטות):**

**{**

**"request\_id": "123456",**

**"timestamp": "2024-10-15T10:00:00Z",**

**"public\_key": "abc123xyz",**

**"hash": "d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2"**

**}**

#### 2. פרוטוקול העבודה

הבקשה מתבצעת על גבי פרוטוקול TCP, שהוא פרוטוקול מבוסס חיבור המספק שירותי העברת נתונים אמינים ומסודרים. פרוטוקול TCP מתאים להעברת בקשות בדיקה מכמה סיבות:

**מעקב אחרי תגובת הצומת**: כאשר השרת שולח את בקשת הבדיקה, הוא מצפה לתשובה מהצומת בתוך פרק זמן קצר. הצומת שולח תשובה בחזרה, מה שמעיד שהוא פועל באופן תקין ומסוגל לתקשר עם הרשת.

**סימון צומת בעייתי**: אם צומת לא מגיב בזמן לבקשת הבדיקה, השרת מסמן אותו כצומת בעייתי או "חולה". אם הצומת ממשיך להיכשל בבדיקות לאורך זמן, השרת עשוי להוציא אותו מהמסלול הפעיל ולהעדיף צמתים אחרים להעברת חבילות.

#### יתרונות השיטה:

* **מעקב שוטף**: השרת מבצע בדיקות באופן רציף כדי לוודא שכל הצמתים משתתפים בצורה תקינה בתקשורת.
* **גמישות בבדיקות**: ניתן לבצע את הבדיקות בצורה אקראית כדי להבטיח שהצמתים אינם מתכוננים מראש או מזייפים תגובות.
* **זיהוי מהיר של תקלות**: שיטה זו מאפשרת לשרת לאתר צמתים לא זמינים או בעלי תקלות בזמן אמת ולהגיב בהתאם.

#### חסרונות השיטה:

* **עומס על השרת**: שליחת בקשות בדיקה תכופות עלולה להעמיס על השרת, במיוחד ברשת גדולה עם מספר רב של צמתים.
* **שימוש במשאבים של הצמתים**: הצמתים נדרשים להגיב לכל בקשת בדיקה, מה שיכול להקטין את הביצועים הכלליים שלהם כשהם עסוקים גם בהעברת חבילות מידע.

**הסבר הקוד**

**מבנה בקשת הבדיקה**: בקשת הבדיקה הנשלחת מהשרת כוללת מזהה ייחודי (Request ID), חותמת זמן (Timestamp), המפתח הציבורי של השרת (Public Key), ושדה אימות המכיל Hash שנוצר באמצעות אלגוריתם כמו SHA-256.

**פרוטוקול העבודה**: הבקשה נשלחת על גבי פרוטוקול TCP, שהוא פרוטוקול חיבורי המספק אמינות וסדר בהעברת הנתונים. השרת עוקב אחרי תגובת הצומת, ואם הצומת לא מגיב תוך הזמן המוקצב, השרת מסמן אותו כבעייתי. הפונקציה SendHealthCheckRequest() שולחת את הבקשה, והפונקציה ReceiveHealthCheckResponse() מקבלת את תגובת הצומת ומוודאת את תקינותה. אם הצומת לא מגיב, הפונקציה MarkNodeAsProblematic() מופעלת, ובמידת הצורך, RemoveNodeFromNetwork() מסירה את הצומת מהמסלול הפעיל של הרשת.

—------------------------------------------------------------

### התחברות למערכת -

התחברות למערכת ברשת Onion Routing מתבצעת באמצעות תהליך מאובטח שמבטיח שהמשתמשים יכולים לגשת למערכת בצורה בטוחה ואנונימית. אחד המרכיבים המרכזיים בתהליך זה הוא שמירה על הסיסמאות בצורה מאובטחת באמצעות אלגוריתם ההצפנה SHA-256.

#### תהליך התחברות:

1. **הרשמה**: כאשר משתמש חדש נרשם למערכת, הוא מתבקש להזין את הסיסמא שלו. לפני שהסיסמא נשמרת במסד הנתונים, היא עוברת תהליך של Hashing באמצעות SHA-256.
   * **Hashing של הסיסמא**: הסיסמא עוברת דרך פונקציית ה-HASH SHA-256, שמייצרת ערך ייחודי (hash) שאינו ניתן להפיכה. תהליך זה מבטיח שהסיסמא המקורית לא נשמרת במערכת, אלא רק הערך המהודר שלה.
2. **שימור במערכת**: הערך המהודר של הסיסמא נשמר במסד הנתונים, בשדה ייחודי שמוקצה עבור כל משתמש.  
   טבלת המשתמשים במערכת עשויה לכלול את השדות הבאים:
   * **USERNAME** - שם המשתמש (טקסט)
   * **PASSWORD\_HASH** - ערך ה-HASH של הסיסמא (טקסט)
3. **התחברות**: כאשר משתמש מנסה להתחבר, הוא מתבקש להזין את שם המשתמש ואת הסיסמא שלו. תהליך ההתחברות מתבצע כך:
   * **חישוב ה-HASH של הסיסמא המוזנת**: הסיסמא המוזנת על ידי המשתמש עוברת גם היא דרך פונקציית SHA-256 ליצירת ערך ה-HASH שלה.
   * **השוואת ערכים**: השרת משווה בין הערך המהודר שהתקבל מהסיסמא המוזנת לבין הערך המהודר ששמור במסד הנתונים. אם הערכים תואמים, המשתמש מאומת ומקבל גישה למערכת.

#### יתרונות השיטה:

* **אבטחת סיסמאות**: שימוש ב-SHA-256 מבטיח שהסיסמאות נשמרות בצורה מאובטחת, ללא חשש להדלפה של הסיסמאות המקוריות.
* **עמידות בפני מתקפות**: הערך המהודר שמתקבל קשה לשחזור, מה שמקשה על תוקפים לגלות את הסיסמאות של המשתמשים במקרה של פרצת אבטחה.

#### חסרונות השיטה:

* **התקפה של Rainbow Tables**: על מנת למנוע מתקפות כאלו, מומלץ להשתמש בטכניקות נוספות כמו Salt (ערך אקראי המוסף לסיסמא לפני החישוב) כדי למנוע מהתוקף להשתמש בטבלאות מוכנות מראש.
* **עיבוד נוסף**: חישוב ה-HASH דורש זמן חישוב נוסף, אך זהו מחיר סביר יחסית לאבטחת המידע.

#### מסד נתונים:

על מנת לשמור את המידע על המשתמשים, ניתן להשתמש במסד נתונים כמו SQLite, אשר יאפשר ניהול של המשתמשים והסיסמאות המהודרות בצורה נוחה ויעילה.

**הסבר קוד**

**הרשמה**: המשתמש מזין סיסמא, והיא עוברת **Hashing** באמצעות הפונקציה SHA256HashPassword(). הערך המהודר נשמר במסד הנתונים בשדה PASSWORD\_HASH.

**התחברות**: בעת התחברות, המערכת מחשבת מחדש את ה-Hash ומבצעת השוואה באמצעות הפונקציה ComparePasswordHashes(). אם הערכים תואמים, המשתמש מאומת.

**שיפור אבטחה עם Salt**: ניתן להוסיף **Salt** לסיסמא באמצעות הפונקציה AddSaltToPassword(), מה שמקשה על מתקפות **Rainbow Tables**.

—------------------------------------------------------------

**שימוש ב-Docker ברשת Tor -**

#### מה זה עושה?

השימוש ב-Docker ברשת Tor מאפשר ליצור צמתים נפרדים שמריצים את הקוד בצורה מבודדת. Docker מספק סביבת פיתוח ובדיקות יעילה לניהול הצמתים ולשליטה בהם. המכולות (containers) של Docker עוזרות לבודד כל צומת ולבדוק את תקינותם באופן נפרד תוך שמירה על גמישות בהגדרת התקשורת ביניהם.

#### קלט:

* **קוד התוכנה של צומת Tor :** כל צומת יצטרך קוד שמגדיר את תפקידו (צומת כניסה, אמצע, יציאה).
* **Dockerfile**: קובץ שמגדיר איך לבנות את המכולה – מהי מערכת ההפעלה הבסיסית, אילו ספריות נדרשות ואילו פקודות להריץ.

#### פלט:

* **צמתים מבודדים**: כל צומת יפעל בתוך מכולה נפרדת, מה שמאפשר בידוד מוחלט בין הצמתים השונים והפחתת סיכוני פריצה או תקלה שתשפיע על כל הרשת.
* **רשת צמתים מאובטחת ומנוהלת**: Docker מאפשר לנהל את הרשת בקלות, להריץ ולהפסיק כל צומת בנפרד, וכן לשכפל ולהרחיב את הרשת על פי הצורך.

#### איך זה עובד?

1. **בניית Dockerfile לכל צומת**: Dockerfile לכל צומת שמרכיב את רשת ה-Tor. הקובץ הזה כולל הוראות ל-Docker איך להקים את הסביבה המתאימה, כולל התקנה של תלותים, קבצים, וקביעת תצורה לצומת.
2. **הרצת המכולות**: כל מכולה רצה בנפרד ומבצעת את תפקידה בצומת Tor שהוגדר לה, תוך כדי בידוד משאר המכולות. מכולות Docker נועדו להיות קלות משקל ולכן ניתן להריץ כמה מהן במקביל, ללא תלות במערכת ההפעלה.
3. **חיבור לצמתים אחרים**: המכולות מתקשרות ביניהן ליצירת רשת Tor מבוזרת, כאשר כל צומת מנהל את החלק שלו בניתוב והצפנת התעבורה. אם אחד מהצמתים נופל, ניתן להפעילו מחדש בקלות באמצעות Docker, מבלי לפגוע ברשת כולה.

#### יתרונות:

* **בידוד ובטיחות**: כל צומת Tor פועל בסביבה מבודדת, כך שאם אחד הצמתים נפרץ או נתקל בבעיה, הוא לא ישפיע על שאר הצמתים או המערכת כולה.
* **גמישות בניהול**: ניתן להפעיל, להפסיק או להוסיף צמתים בקלות ובמהירות בהתאם לצורך, מה שמאפשר גמישות רבה.
* **שכפול מהיר**: אפשר לשכפל מכולות בקלות ולהרחיב את הרשת או להוסיף עוד צמתים, מה שתורם ליכולת ההתרחבות של הרשת.

**קונטיינר (Container) -**

קונטיינר הוא יחידת תוכנה קלת משקל ובידודית שמכילה את כל מה שהיישום צריך כדי לרוץ – קוד, תלויות, קבצים, וספריות – מבלי להיות תלוי במערכת ההפעלה המארחת. הקונטיינר רץ בסביבה מבודדת משאר הקונטיינרים ומערכת ההפעלה, ולכן הוא מאפשר להריץ מספר יישומים על אותו מחשב או שרת מבלי ליצור התנגשויות ביניהם.

#### איך קונטיינר שונה ממכונה וירטואלית?

* **משקל קל**: בניגוד למכונות וירטואליות שמריצות מערכת הפעלה שלמה, קונטיינר חולק את ליבת מערכת ההפעלה עם המארח, מה שהופך אותו למהיר וקל יותר.
* **בידוד אפקטיבי**: למרות שהקונטיינרים חולקים את ליבת מערכת ההפעלה, הם עדיין מבודדים זה מזה כך שאם אחד נופל או נפרץ, שאר הקונטיינרים לא יושפעו.

#### איך זה עובד?

כאשר מריצים קונטיינר, Docker יוצר סביבה וירטואלית בה הקוד יכול לפעול בצורה עצמאית, כאילו הוא רץ על שרת משלו. הקונטיינר מכיל את הקוד, כל התלויות שהוא צריך (כגון ספריות וקבצים), ומערכת הפעלה מינימלית הנדרשת להרצה.

### הקשר בין שרת הדיירקטורי לבין הקונטיינרים (Docker):

* **פרסום מידע הצמתים**: כל צומת שמריצים בתוך קונטיינר (Docker) צריך לפרסם את עצמו לשרת הדיירקטורי כדי שיידעו עליו ויוכלו להשתמש בו. הקונטיינרים שרצים בעצם מהווים את הצמתים עצמם, ודרך Docker אפשר לנהל אותם בקלות.
* **עדכון דינמי**: אם מוסיפים או משכפלים צמתים חדשים בעזרת Docker, שרת הדיירקטורי צריך להתעדכן במידע על הצמתים החדשים, כך שהרשת תוכל לדעת עליהם.
* **חיבור בין צמתים**: כאשר משתמש ברשת רוצה להתחבר לצומת, הוא ישתמש במידע שמתקבל משרת הדיירקטורי כדי לזהות אילו צמתים קיימים, מה הכתובות שלהם, ואיזה צומת מתאים לנתיב האנונימי המבוקש.

### איך זה עובד?

1. **Docker להרצת צמתים**: מריצים את הצמתים כמכולות Docker. כל מכולה מייצגת צומת שמריץ את קוד ה-Tor.
2. **שרת הדיירקטורי**: שרת הדיירקטורי מקבל פרטים מכל צומת שנוצר, ומרכז את כל המידע על הצמתים הזמינים ברשת.
3. **תקשורת בין הצמתים**: המשתמשים שצריכים ליצור מסלול ברשת Tor (או צמתים אחרים) יקבלו את המידע על הצמתים, ויוכלו ליצור קשר עם הצמתים דרך רשת Docker.

Docker מבטיח שכל צומת יפעל באופן מבודד ויאפשר לשלוט על רשת הצמתים בקלות, בעוד ששרת הדיירקטורי הוא ה"מוח" שמנהל את הקשר בין הצמתים ומעדכן אותם בזמן אמת.

—------------------------------------------------------------

### העברת סרטונים ברשת Tor -

#### מה זה עושה?

הפיצ'ר הזה מאפשר למשתמשים להעלות, לשתף ולצפות בסרטונים דרך פלטפורמה, תוך כדי שימוש ברשת Tor כדי להבטיח אבטחה ואנונימיות. הפלטפורמה תאפשר למשתמשים להתנהל עם סרטונים בצורה מאובטחת, מבלי לחשוף את זהותם. באמצעות טכנולוגיות הצפנה והפצה מבוזרת, הפיצ'ר מבטיח שהסרטונים מועברים מבלי לחשוף את זהות השולח.

#### קלט:

* **סרטון**: קובץ הווידאו שהמשתמש רוצה להעביר.
* **תיאור הסרטון**: טקסט שמסביר את תוכן הסרטון.
* **תגיות**: מילות מפתח שיכולות לעזור למשתמשים למצוא את הסרטון בקלות.
* **עדיפויות פרטיות**: אפשרויות לקביעת רמות פרטיות לסרטון (למשל, ציבורי, פרטי, או מוגבל למשתמשים מסוימים).

#### פלט:

* **אישור העברה**: הודעה על כך שהסרטון הועבר בהצלחה לצומת היעד.
* **דף סרטון**: דף שיציג את הסרטון עם תיאור

#### איך זה עובד?

1. **העלאת הסרטון**:
   * המשתמש בוחר את הסרטון שהוא רוצה להעביר
   * המשתמש מעלה סרטון דרך הממשק של הפלטפורמה.
   * המערכת תבצע הצפנה על הסרטון ותשמור אותו בשרתים מבוזרים ברשת Tor.
2. **מציאת הצומת הנכון**:
   * המערכת פונה לשרת הדיירקטורי כדי למצוא את הצומת שאליו יש להעביר את הסרטון.
   * המידע כולל את כתובת ה-IP ואת המפתח הציבורי של הצומת.
3. **העברת הסרטון**:
   * הסרטון מועבר דרך רשת Tor במסלול מבוזר.
   * כל צומת בדרך מפנה את המידע לצומת הבא, תוך שמירה על הצפנה של המידע בכל שלב.
4. **אישור לסיום התהליך**:
   * לאחר שהסרטון הועבר בהצלחה, המערכת שולחת הודעה למשתמש לאשר את סיום ההעברה.

5. **פרסום הסרטון**:

* לאחר ההעלאה, הסרטון יתפרסם בפלטפורמה עם תיאור ותגיות.
* כל סרטון יקבל כתובת ייחודית שיאפשרת למשתמשים לגשת אליו.

6. **צפייה בסרטונים**:

* משתמשים יוכלו לחפש ולצפות בסרטונים לפי תגיות או שמות משתמשים.
* הסרטונים ייטענו דרך רשת Tor, תוך כדי שמירה על אנונימיות המשתמשים.

#### יתרונות:

* **אבטחה גבוהה**: הסרטונים מועברים בצורה מוצפנת ומבוזרת כלומר הרשת מספקת למשתמשים הגנה על זהותם בזמן שהם מעלים או צופים בסרטונים.
* **אנונימיות**: השימוש ברשת Tor מבטיח שהמשתמשים נשארים אנונימיים במהלך העברת הסרטונים.
* **נגישות מבוזרת**: המערכת מאפשרת למשתמשים לגשת לתוכן מכל מקום, כל עוד יש להם חיבור לרשת Tor.

—------------------------------------------------------------

**טכנולוגיות**

| **פיצ'ר** | **טכנולוגיות ושפות תכנות** | **משאבים נדרשים** |
| --- | --- | --- |
| שכבות ההצפנה | טכנולוגית AES על ידי שפת תכנות C++ | מעבד |
| הצפנת AES | שפת תכנות c++ | סביבת עבודה |
| הצמתים וחיבורם לרשת |  |  |
| הצומת האחרון במסלול |  |  |
| בחירת המסלול דרכו תעבור החבילה | אלגוריתם הבנוי על C++ | סביבת עבודה |
| העברת המידע דרך הצמתים | TCP | מחשב ( חיבור לרשת ) |
| ניהול הצמתים |  |  |
| מימוש פרוטוקול TCP | socket raw \ c++ | סביבת עבודה (VS) \ חיבור לרשת |
| העברת קבצים ברשת TOR | segments |  |
| הצפנה אסימטרית עם מפתחות סימטריים |  |  |
| שמירת מפתחות פומביים של הצמתים |  |  |
| בדיקת תקינות צמתים |  |  |
| התחברות למערכת |  |  |
| שימוש ב-Docker ברשת Tor |  | שימוש בתוכנת dockers |
| העברת סרטונים ברשת Tor |  |  |

**ספריות ותפקידם :  
  
  
 Sink:**

* **תפקיד**: צומת מרכזי (Sink) ברשת החיישנים האלחוטיים (WSN). זה הצומת שאליו נשלחים כל הנתונים שנאספו מהצמתים האחרים ברשת.
* **פונקציות עיקריות**:
  + קבלת מפתחות ציבוריים מכל צומת (Handshake).
  + בניית הודעות ה-Onion ושליחתן לצמתים ברשת.
  + ניהול מסלול ההודעה והצמתים שדרכם היא תעבור.
  + מעקב אחר הצלחת ההעברה ותיקון בעיות בהעברת הודעות.

**SensorNode :**

* **תפקיד**: צומת חיישן (Node) ברשת החיישנים. תפקידו לקלוט נתונים מהסביבה ולהעביר את המידע לצומת הבא או לצומת המרכזי (Sink).
* **פונקציות עיקריות**:
  + קבלת הודעות Onion, פענוח השכבה המתאימה, וביצוע משימות הנדרשות (כמו חישוב או הצפנה מחדש).
  + שליחת הודעת ה-Onion לצומת הבא במסלול.
  + תקשורת עם הצומת המרכזי (Sink) כדי לרשום את המפתח הציבורי שלו בתחילת התהליך.

**OnionManager :**

* **תפקיד**: אחראי על ניהול תהליך ההצפנה והפענוח של שכבות ה-Onion.
* **פונקציות עיקריות**:
  + יצירת "ראש ה-Onion", הכולל שכבות הצפנה של ההודעה.
  + ניהול ההצפנה והפענוח של כל שכבה בהודעה.
  + שימוש בספריית Libsodium לביצוע ההצפנה על בסיס מפתחות ציבוריים-פרטיים (Curve25519).

**OnionValidator :**

* **תפקיד**: מעקב אחרי הצלחת העברת הודעות ה-Onion בין הצמתים.
* **פונקציות עיקריות**:
  + בדיקת הגעת ההודעה לצומת הבא במסלול.
  + במידה ויש כשל בהעברה (למשל, ההודעה לא הגיעה בזמן שנקבע), המערכת שולחת הודעה חדשה דרך המחלקה הזו.

**WsnConstructor :**

* **תפקיד**: בונה את הרשת (הטופולוגיה) ואת ההתקנים הנדרשים להפעלת רשת החיישנים.
* **פונקציות עיקריות**:
  + יצירת הצמתים ברשת והתקנת היישומים הנדרשים עליהם.
  + בניית הטופולוגיה (מיקומי הצמתים, תדרי התקשורת וכו').
  + התקנת הפרוטוקולים הנדרשים לניתוב התקשורת ברשת.

**OutputManage :**

* תפקיד: ניהול הפלט של הסימולציה.
* פונקציות עיקריות:
  + רישום תוצאות הסימולציה (כמו זמני נסיעה של הודעות, מסלול ההודעות, שגיאות וכו') בקובצי פלט.
  + כתיבה למסך (stdout) וגם לקובץ נפרד לצורך ניתוח תוצאות הסימולציה.

**סיכום תפקיד המחלקות:**

* **Sink** (שרת דירקטורי) הוא המנהל המרכזי שאחראי על שליחת ההודעות וניהול המסלולים.
* **SensorNode**  - הם הצמתים שאוספים נתונים ומעבדים את ההודעות.
* **OnionManager** - מטפל בהצפנה ובפענוח של שכבות ההודעות.
* **OnionValidator**  - מוודא שההודעות מגיעות ליעדן ומנהלת בעיות בהעברה.
* **WsnConstructor**  - בונה את הרשת.
* **OutputManager**  - מטפל בתוצאות ובפלט של הסימולציה.

**מבנה בסיס נתונים**

בפרויקט נממש את מסד הנתונים באמצעות Access DB מסוג sqlite3 לשרת דירקטורי

טבלה של צמתים ברשת :

* שדה PORT - מסוג Integer
* שדה IP - מסוג Integer
* שדה PUBLIC\_KEY - מסוג Integer



2. 
3. **פרק 3: ארכיטקטורה**
4. **מבט על**

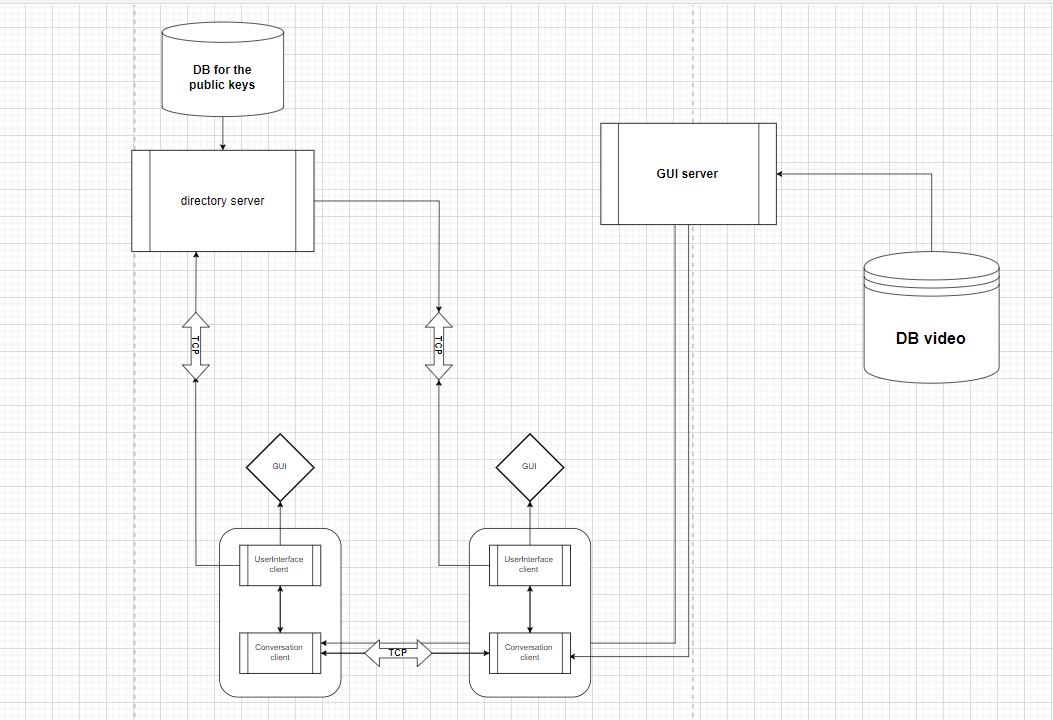
כשעבדנו על תוכנות, חילקנו אותן לפונקציות – לכל פונקציה תפקיד ותחום אחריות משלה. כשעבדנו על תוכנות גדולות, חילקנו אותן למחלקות – לכל מחלקה תפקיד ואחריות משלה. ועכשיו – נחלק את הפרויקט שלנו לרכיבים – לכל רכיב תפקיד ואחריות משלו. דוגמאות לרכיבים: שרת, לקוח ומסד נתונים.

כאן יש לפרט את הרכיבים העיקריים שמרכיבים את מימוש הפרויקט. אין צורך להיכנס לדקויות או למקרי קצה, אלא לתת תמונה כוללת של החלקים העיקריים במערכת.

ראשית כל יש לשים כאן דיאגרמה המתארת את **הרכיבים העיקריים** ו**הקשרים ביניהם**. את הדיאגרמה ניתן ליצור במגוון דרכים - החל מסריקה של שרטוט שציירתם בעצמכם, דרך Powerpoint ועד לשירותי רשת חינמיים כמו [moqups.com](about:blank) או [Draw.io](http://draw.io/).

לאחר מכן יש להסביר בקצרה עבור כל רכיב לגבי תפקידו ותחום האחריות שלו ולאילו רכיבים אחרים הוא מתממשק.

💖 שימו לב: יתכן שהפרויקט שלכם יחולק למספר תוכנות/אפליקציות שונות, כאשר לכל אחת מהן קיימים רכיבים משלה. עליכם לפרט על כל הרכיבים בפרוייקט שלכם.



**רכיבים**

**Sink.h**

רכיב **Sink** שעושה: מנהל את התקשורת המרכזית ברשת החיישנים (WSN), מקבל נתונים מהצמתים, שולח הודעות Onion, ומנהל את מסלול ההודעה.

* תת רכיב **Handshake()** שעושה: מבצע קבלת מפתחות ציבוריים מכל צומת בתהליך התחלת התקשורת.
* תת רכיב **SelectRoute()** שעושה: בוחר את המסלול שבו ההודעה תעבור בין הצמתים.
* תת רכיב **PrepareOnion()** שעושה: בונה את הודעת ה-Onion על ידי יצירת שכבות הצפנה שונות.
* תת רכיב **SendSegment()** שעושה: שולח את הודעת ה-Onion לצמתים שנבחרו במסלול.

### SensorNode.h

רכיב **SensorNode** שעושה: פועל כצומת חיישן, קולט ומעבד הודעות Onion ומעביר אותן הלאה בצורתן המעובדת.

* תת רכיב **OnionReceived()** שעושה: קולט הודעת Onion ומעבד אותה.
* תת רכיב **ProcessOnionHead()** שעושה: פענוח ראש ה-Onion (השכבה המתאימה) וביצוע המשימות הנדרשות.
* תת רכיב **ProcessOnionBody()** שעושה: עיבוד גוף ההודעה לאחר פענוח ראש ה-Onion.
* תת רכיב **SendSegment()** שעושה: שולח את הודעת ה-Onion לצומת הבא במסלול שנבחר.
* תת רכיב **CheckOnionReceived()** שעושה: מאמת את קבלת ההודעה.
* תת רכיב **DisableNode()** שעושה: משבית את הצומת במקרה של תקלה.
* תת רכיב **ActivateNode()** שעושה: מפעיל מחדש את הצומת לאחר השבתה.

### 

### OnionManager.h

רכיב **OnionManager** שעושה: מטפל בתהליך ההצפנה והפענוח של הודעות Onion לאורך המסלול.

* תת רכיב **EncryptLayer()** שעושה: מבצע הצפנה של שכבה בהודעת ה-Onion לפי המפתחות המתאימים.
* תת רכיב **DecryptLayer()** שעושה: מבצע פענוח של שכבה מההודעה, לפי המפתחות הציבוריים-פרטיים של הצומת.

### OnionValidator.h

רכיב **OnionValidator** שעושה: מוודא את תקינות התקשורת של הודעות Onion במהלך העברתן בצמתים.

* תת רכיב **OnionReceived()** שעושה: בודק אם הודעת ה-Onion הגיעה לצומת היעד.
* תת רכיב **CheckOnionReceived()** שעושה: מאמת את הגעת ההודעה בזמן שנקבע ומנהל הודעות במקרה של כשל.

### WsnConstructor.h

רכיב **WsnConstructor** שעושה: בונה את רשת החיישנים, את הצמתים ואת הפרוטוקולים הנדרשים לפעולתה.

* תת רכיב **BuildNetwork()** שעושה: מייצר את הצמתים ברשת, מתקין יישומים, ובונה את טופולוגיית הרשת.

### OutputManager.h

רכיב **OutputManager** שעושה: מטפל ביצירת הפלט ובתוצאות הסימולציה.

* תת רכיב **WriteToFile()** שעושה: כותב את תוצאות הסימולציה (כמו נתוני מסלול ושגיאות) לקובץ פלט לניתוח.
* תת רכיב **WriteToStdout()** שעושה: כותב את תוצאות הסימולציה למסך לצורך ניתוח בזמן אמת.

### מבנה מערכת:

#### Sink (שרת דירקטורי)

* **לוגיקה**: אחראי על בחירת מסלול, הכנת הודעות ושליחת הודעות ה-Onion לצמתים ברשת.
* **תקשורת עם SensorNodes**: מבצע אותנטיקציה של הצמתים, מקבל מפתחות ציבוריים ושולח מקטעים מוצפנים של הודעות Onion.
* **ניהול מסלולים**: עוקב אחר מסלולי ההודעה ברשת, מוודא הצלחה בהעברת ההודעות, ומבצע התאמות במסלול במידה וישנן בעיות.

#### SensorNodes (צמתי חיישן)

* **לוגיקה**: מקבל הודעות Onion, מפענח את השכבה המתאימה ושולח את ההודעה הלאה או מעבד אותה אם הוא היעד הסופי.
* **תקשורת עם ה-Sink**: מבצע החלפת מפתחות ורישום, ומוודא שהצומת מוכן להשתתף בהעברת ההודעות המוצפנות.
* **עיבוד Onion**: מבצע פענוח או הצפנה מחדש של שכבות Onion בהתאם לשלב הנוכחי של הניתוב.
* **ניהול תקלות**: עוקב אחר הצלחת עיבוד ההודעה, ואם יש כשל, מבצע שידור מחדש או שולח התראה ל-Sink.

#### OnionManager (ניהול שכבות ההצפנה)

* **לוגיקה**: מנהל את תהליך ההצפנה והפענוח של שכבות ה-Onion.
* **תקשורת עם Libsodium**: משתמש בספריית Libsodium לביצוע הצפנה באמצעות מפתחות ציבוריים-פרטיים (Curve25519).
* **ניהול שכבות**: מטפל ביצירת מבנה ה-Onion ומוודא שכל שכבה מוצפנת בצורה מאובטחת להעברתה דרך הצמתים.

#### OnionValidator (ולידציה של העברת הודעות Onion)

* **לוגיקה**: עוקב אחרי הצלחת העברת ההודעות לצמתים הנכונים ברשת.
* **זיהוי כשלים**: שולח הודעות שגיאה או מחדש את ההעברה במקרה של כשל בהעברת הודעה לצומת הבא במסלול.

#### WsnConstructor (בנייה וניהול של הרשת)

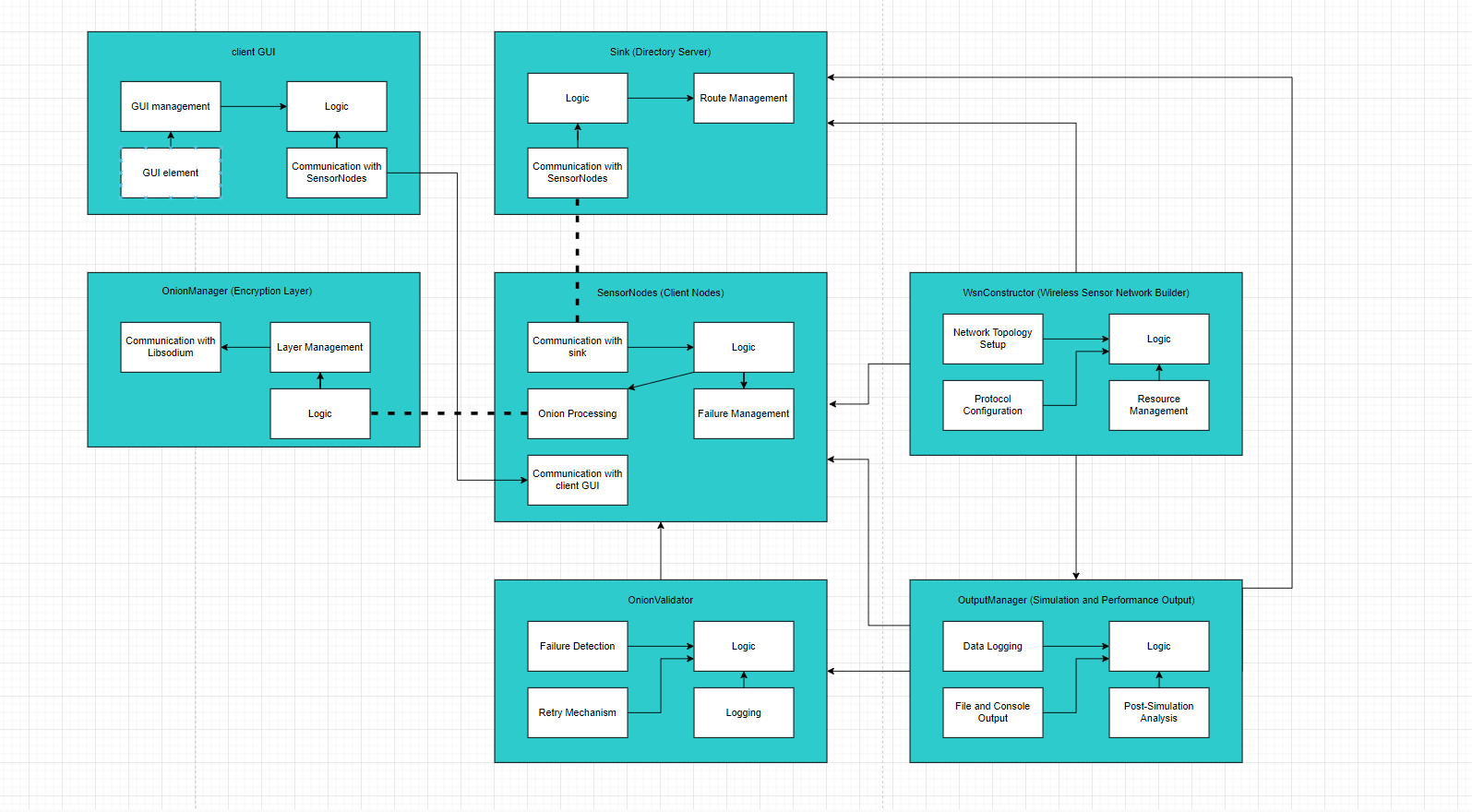
* **לוגיקה**: בונה את הטופולוגיה של הרשת ויוצר את הצמתים הנדרשים.
* **התקנה ופריסה**: מבצע התקנה של התוכנות הנדרשות על הצמתים ובונה את התקשורת בין הצמתים ברשת.

#### OutputManager (ניהול פלט הסימולציה)

* **לוגיקה**: מתעד את תוצאות הסימולציה כמו זמני נסיעה של הודעות, שגיאות, ומסלולי ההודעות.
* **כתיבת פלט**: רושם את התוצאות לקבצי פלט ולמסך לצורך ניתוח.

### תפקידים עיקריים:

* **Sink (שרת דירקטורי)**: מנהל ראשי של שליחת הודעות וניהול מסלולים.
* **SensorNodes (צמתים)**: אוספים נתונים ומעבדים את ההודעות.
* **OnionManager**: מטפל בהצפנה ופענוח של שכבות ההודעות.
* **OnionValidator**: מוודא שהודעות מגיעות ליעדן ומנהל שגיאות בהעברה.
* **WsnConstructor**: בונה את רשת הצמתים.
* **OutputManager**: מטפל בפלט ובתוצאות הסימולציה.



### OnionManager -

**תפקיד רכיב הלוגיקה:**רכיב הלוגיקה ב-OnionManager אחראי על תיאום תהליך ההצפנה של שכבות הנתונים, כלומר, הוא מגדיר מתי ואיך לעטוף נתונים בשכבות הצפנה מרובות ומנהל את התלויות בין המשימות השונות.

**ניהול זרימת מידע ותלויות -**

1. **שליטה בזרימת המידע:**
   * **זיהוי משימות:** רכיב הלוגיקה מזהה את סדר הפעולות הנדרש עבור כל חבילה, הכולל הצפנת שכבות, סדר ריליי (Relay) והעברה סופית.
   * **אסטרטגיית ניתוב:** מגדיר את מספר השכבות הדרושות בהתבסס על מספר הצמתים במסלול, ומקצה לכל שכבה מפתח הצפנה ייחודי.
   * **פיצול נתונים:** מחלק את הנתונים לחבילות במידת הצורך, ומקצה לכל חבילה תהליך הצפנה לפי המסלול.
2. **ניהול תלויות:**
   * **תלויות בשכבות הצפנה:** מגדיר את היחס בין כל שכבת הצפנה כך שכל שכבה מיושמת לפי סדר מסוים.
     + לדוגמה, מבטיח שהצפנה של *שכבה 3* לא תתחיל עד לסיום ההצפנה של *שכבה 2*, וכן הלאה.
   * **ניהול תלות ב-Libsodium:** שולט במתי יש לקרוא ל-Libsodium לצורך הפונקציות הקריפטוגרפיות, ומסנכרן כל קריאה עם סדר השכבות.
     + מטפל ביצירת מפתחות והחלפתם עבור כל שכבה כך שכל צומת במסלול יקבל את המפתח הנכון לפענוח.
3. **טיפול בתקלות ותסריטי גיבוי:**
   * **אימות שכבות:** אחרי כל שכבה של הצפנה, רכיב הלוגיקה בודק שהשכבה יושמה בצורה תקינה.
   * **פרוטוקול תקלות:** אם שכבה נכשלה בהצפנה או בפענוח, מבצע ניסיון חוזר עם מפתחות גיבוי או מבצע הצפנה מחדש מהשכבה האחרונה שהוצפנה בהצלחה.
4. **ניהול תור עיבוד שכבות:**
   * **תור שכבות:** מנהל תור שבו כל חבילה עוברת דרך השכבות בזו אחר זו.
   * **עיבוד מקבילי:** אם התשתית מאפשרת, רכיב הלוגיקה יכול לנהל מספר חבילות בתור, ולעבד אותן בהצפנה בזו אחר זו או במקביל (בהתאם למשאבים).
5. **תקשורת עם תתי רכיבים:**
   * **תקשורת עם ניהול שכבות:** שולח פקודות לניהול שכבות להתחיל לעטוף חבילה בשכבות או להתחיל בפענוח השכבות אם הוא מקבל נתונים.
   * **תיאום עם Libsodium:** קורא ל-Libsodium עבור כל שכבת הצפנה, מעביר את הנתונים והמפתחות הנדרשים ומקבל נתונים מוצפנים.
   * **דיווח:** שולח עדכוני סטטוס וכל תקלה קריטית (כגון כשל בניסיון הצפנה) ליומן התקלות או ל-OnionValidator לפעולה נוספת.

### דוגמה לזרימה בתוך רכיב הלוגיקה של OnionManager

1. **קבלת חבילת נתונים** **—>** זיהוי מסלול ומספר השכבות **—>** הצבה בתור לעיבוד שכבות
2. **עבור כל שכבה:**
   * קריאה ל-**Libsodium** להצפנה, קבלת חבילה מוצפנת.
   * אם הצליחה, העברה לשכבה הבאה.
   * אם נכשלה, ניסיון חוזר או העברת התרעה ל-OnionValidator.
3. **חבילה סופית:** כאשר כל השכבות מוצפנות, מסמן את החבילה כמוכנה לשליחה דרך הצמתים.

### OnionValidator -

**תפקיד רכיב הלוגיקה:**רכיב הלוגיקה בתוך OnionValidator אחראי על ניהול כל הפעילויות הקשורות לשלמות הנתונים, ניהול תקלות ואסטרטגיות לניסיונות חוזרים. רכיב זה מקבל עדכונים מזיהוי תקלות, מנגנון הניסיונות החוזרים ומרישום, ומשלב את התגובות שלהם בתהליך אחיד לטיפול בתקלות.

#### 1. זיהוי תקלות —> רכיב הלוגיקה

* **מטרה של זיהוי תקלות:** מזהה בעיות בהעברת נתונים, כמו שחיתות, אובדן נתונים או שינויים לא צפויים.
* **אינטראקציה עם רכיב הלוגיקה:** כאשר מזוהה תקלה, זיהוי תקלות מדווח על פרטי התקלה לרכיב הלוגיקה.
  + **תגובה של רכיב הלוגיקה:** בהתבסס על סוג התקלה, רכיב הלוגיקה מפעיל את מנגנון הניסיונות החוזרים לשלוח את הנתונים שוב או רושם את התקלה וממשיך לפקח על תקלות חדשות.

#### 2. מנגנון ניסיונות חוזרים —> רכיב הלוגיקה

* **מטרה של מנגנון ניסיונות חוזרים:** מנסה לשלוח שוב את הנתונים באופן אוטומטי כאשר מזוהה תקלה בהעברה או באימות.
* **אינטראקציה עם רכיב הלוגיקה:** מנגנון הניסיונות החוזרים מדווח על כל ניסיון חזרה לרכיב הלוגיקה, כולל מצב הניסיון ומספר הניסיונות שבוצעו.
  + **תגובה של רכיב הלוגיקה:** אם הניסיונות נכשלו באופן עקבי, רכיב הלוגיקה מחליט להפסיק את הניסיונות החוזרים ומסמן את התקלה לרישום.

#### 3. רישום —> רכיב הלוגיקה

* **מטרה של רישום:** רושם את כל האירועים, התקלות וניסיונות החזרה למטרות בקרה וניתוח.
* **אינטראקציה עם רכיב הלוגיקה:** רכיב הרישום שולח דיווח מצב על כל תקלה או ניסיון חזרה שנרשם לרכיב הלוגיקה.
  + **תגובה של רכיב הלוגיקה:** רכיב הלוגיקה משתמש במידע הזה לניתוח יציבות המערכת ומגמות ביצועים, מספק תובנות לגבי נקודות כשל נפוצות או שיפורים נדרשים.

### דוגמת תהליך ברכיב הלוגיקה של OnionValidator

1. **זיהוי תקלה:** זיהוי תקלות מזהה בעיה ומדווח לרכיב הלוגיקה.
2. **ניסיון חוזר:** רכיב הלוגיקה מפעיל את מנגנון הניסיונות החוזרים בניסיון לשלוח את המידע מחדש.
   * **אם הניסיון נכשל:** לאחר מספר מוגדר של ניסיונות, מנגנון הניסיונות החוזרים מדווח לרכיב הלוגיקה שכל הניסיונות נכשלו.
3. **רישום תקלה:** רכיב הלוגיקה מנחה את רכיב הרישום לתעד את הכשל לצורך ניתוח עתידי.
4. **לולאת משוב:** המידע שנשמר על ידי הרישום משמש בהמשך את רכיב הלוגיקה לשיפור אסטרטגיות הניסיונות החוזרים או לזיהוי תבניות למניעת תקלות מראש.

### WSNConstructor - רכיב הלוגיקה עם תתי רכיבים

**תפקיד רכיב הלוגיקה:**רכיב הלוגיקה בתוך WSNConstructor אחראי על תכנון מבנה הרשת האלחוטית, קביעת הפרוטוקולים לניהול התקשורת, וניהול המשאבים כך שכל רכיב ברשת יוכל לתפקד בצורה מיטבית. רכיב זה מקבל מידע ועדכונים על ההגדרות מטופולוגיית הרשת, מהגדרות הפרוטוקול ומהמשאבים, ומשלב אותם לפעולה מאוחדת ויעילה.

#### 1. הגדרת טופולוגיית הרשת —> רכיב הלוגיקה

* **מטרה:** קובעת איך הצמתים ברשת יהיו מחוברים זה לזה, ואיך הנתונים יזרמו בין הצמתים.
* **אינטראקציה עם רכיב הלוגיקה:** רכיב הטופולוגיה שולח את מבנה הרשת לרכיב הלוגיקה.
  + **תגובה של רכיב הלוגיקה:** בוחן את מבנה החיבורים שנקבע ובודק אם הוא מתאים להקצאות הפרוטוקול ולמשאבים.

#### 2. קביעת הפרוטוקול —> רכיב הלוגיקה

* **מטרה:** מגדיר את כללי התקשורת בין הצמתים, כגון סדר החלפת הודעות ושיטת סנכרון בין רכיבים.
* **אינטראקציה עם רכיב הלוגיקה:** קביעת הפרוטוקול מעבירה לרכיב הלוגיקה את ההנחיות לגבי תזמון התקשורת ושיטות הקישור.
  + **תגובה של רכיב הלוגיקה:** משתמש במידע זה כדי ליישם את הפרוטוקול על טופולוגיית הרשת שנקבעה ומוודא שההגדרות תואמות.

#### 3. ניהול משאבים —> רכיב הלוגיקה

* **מטרה:** להקצות את המשאבים בצורה יעילה כך שלכל צומת יהיה מספיק זיכרון, כוח עיבוד, ואנרגיה לפעול.
* **אינטראקציה עם רכיב הלוגיקה:** רכיב ניהול המשאבים שולח מידע על הקצאת המשאבים לרכיב הלוגיקה.
  + **תגובה של רכיב הלוגיקה:** דואג שהמשאבים מוקצים בצורה שתתמוך במבנה ובפרוטוקול שנקבעו, ומוודא שלא יהיו עומסי יתר על צמתים מסוימים.

### דוגמת תהליך ברכיב הלוגיקה של WSNConstructor

1. **הקצאת טופולוגיה:** הטופולוגיה מוגדרת לפי מבנה הרשת והצמתים **—>** רכיב הלוגיקה מאמת את המבנה.
2. **קביעת פרוטוקול:** רכיב הפרוטוקול שולח הוראות תקשורת **—>** רכיב הלוגיקה מוודא שהן תואמות את הטופולוגיה.
3. **ניהול משאבים:** ניהול המשאבים מוודא שיש מספיק כוח עיבוד ואנרגיה לכל צומת **—>** רכיב הלוגיקה משלב את כל המידע ומתחיל בתהליך הבנייה הסופי של הרשת.

### Output Manager - רכיב הלוגיקה ותפקודו

**תפקיד רכיב הלוגיקה:**רכיב הלוגיקה בתוך Output Manager אחראי על ניהול ואיחוד המידע שהתקבל מהסימולציה, קביעת תהליך הרישום, הפקת דוחות לקובץ או לקונסולה, וביצוע ניתוחים וסטטיסטיקות לאחר הסימולציה. רכיב הלוגיקה מקבל נתונים מכל אחד מתתי הרכיבים, מסנן, מאגד ומנתח אותם כדי להפיק תובנות משמעותיות.

#### 1. רישום נתונים (Data Logging) —> רכיב הלוגיקה

* **מטרה:** לאסוף את כל האירועים המרכזיים והתוצאות שהתרחשו במהלך הסימולציה.
* **אינטראקציה עם רכיב הלוגיקה:** רישום הנתונים שולח את הנתונים שנאספו לרכיב הלוגיקה להמשך עיבוד.
  + **תגובה של רכיב הלוגיקה:** מסנן את הנתונים, ממיין אותם לפי קטגוריות רלוונטיות, ומתעדף אירועים חשובים.

#### 2. יציאה לקובץ ולקונסולה (File and Console Output) —> רכיב הלוגיקה

* **מטרה:** להציג את תוצאות הסימולציה בצורה קריאה לקונסולה, ולשמור את הנתונים לדוחות בקובץ.
* **אינטראקציה עם רכיב הלוגיקה:** רכיב היציאה שולח בקשות לרכיב הלוגיקה כדי להחליט אילו נתונים יש להציג ישירות בקונסולה ואילו יש לשמור בקובץ לדוחות.
  + **תגובה של רכיב הלוגיקה:** מחליט על הנתונים שיוצגו בזמן אמת, מסנן מידע רלוונטי לדוחות בקובץ, ומבצע עיבוד ראשוני אם נדרש.

#### 3. ניתוח לאחר הרצה (Post Run Analysis) —> רכיב הלוגיקה

* **מטרה:** לבצע ניתוחים סטטיסטיים, לזהות דפוסים ולסכם את ביצועי המערכת לאחר סיום ההרצה.
* **אינטראקציה עם רכיב הלוגיקה:** הניתוח לאחר הסימולציה מעביר נתונים סטטיסטיים וחישובים מסכמים לרכיב הלוגיקה.
  + **תגובה של רכיב הלוגיקה:** מאחד את התוצאות, יוצר סטטיסטיקות מסכמות, ומפיק תובנות מהנתונים כדי לספק סיכום ביצועים מלא.

### דוגמת תהליך בתוך רכיב הלוגיקה של Output Manager

1. **רישום נתונים:** רכיב הרישום אוסף את כל האירועים המרכזיים מהסימולציה **—>** רכיב הלוגיקה מסנן ומסווג את המידע.
2. **הצגה בזמן אמת:** רכיב היציאה מציג חלק מהמידע לקונסולה ושומר דוחות לקובץ **—>** רכיב הלוגיקה בוחר את הנתונים הרלוונטיים.
3. **ניתוח מסכם:** רכיב הניתוח מבצע חישובים מסכמים **—>** רכיב הלוגיקה מאגד את הנתונים לדוח מסכם על ביצועי הסימולציה.

### Sink - רכיב הלוגיקה ותפקידו

**תפקיד רכיב הלוגיקה:**רכיב הלוגיקה בתוך ה-Sink אחראי על ניהול התקשורת עם הצמתים הסנסוריים (Sensor Nodes) וביצוע תהליכים חשובים כמו ניהול המסלולים לנתונים העוברים דרך הרשת. רכיב זה מרכז את ההודעות המתקבלות מהצמתים, מנתח את המידע, ומקבל החלטות לגבי הנתיב היעיל ביותר להעברת הנתונים.

#### 1. תקשורת עם Sensor Nodes —> רכיב הלוגיקה

* **מטרה:** לתקשר עם הצמתים הסנסוריים, לקבל מהם נתונים ולשלוח הוראות.
* **אינטראקציה עם רכיב הלוגיקה:** רכיב התקשורת מעביר את המידע המתקבל מהצמתים הסנסוריים לרכיב הלוגיקה.
  + **תגובה של רכיב הלוגיקה:** מנתח את המידע המתקבל ומבצע החלטות על סמך המידע, כגון איזה נתיב לנהל ואילו צמתים דורשים תשומת לב מיוחדת.

#### 2. רכיב הלוגיקה —> ניהול מסלולים (Route Management)

* **מטרה:** לקבוע את המסלולים היעילים ביותר להעברת הנתונים דרך הרשת.
* **אינטראקציה עם רכיב הלוגיקה:** רכיב הלוגיקה מעביר את ההמלצות לגבי ניהול המסלולים לרכיב ניהול המסלולים.
  + **תגובה של רכיב ניהול המסלולים:** מבצע את השינויים הנדרשים במבנה המסלול על סמך ההמלצות, כגון שינוי הנתיב או הקצאת צמתים חדשים להעברת נתונים.

### דוגמת תהליך בתוך רכיב הלוגיקה של Sink

1. **איסוף נתונים:** רכיב התקשורת מקבל נתונים מהצמתים הסנסוריים **—>** רכיב הלוגיקה מנתח את המידע המתקבל.
2. **החלטת ניהול מסלול:** רכיב הלוגיקה קובע אילו מסלולים יש לנהל עבור הנתונים המתקבלים **—>** הוא מעביר המלצות לרכיב ניהול המסלולים.
3. **ביצוע שינויים:** רכיב ניהול המסלולים מבצע את השינויים הנדרשים לנתיב הנתונים על סמך ההמלצות מהלוגיקה.

### Client GUI - רכיב הלוגיקה ותפקידו

**תפקיד רכיב הלוגיקה:**רכיב הלוגיקה בתוך Client GUI אחראי על קביעת הדרך בה המידע מוצג למשתמש, כמו גם על ניהול התקשורת עם הצמתים הסנסוריים. רכיב זה מרכז את כל ההגדרות, הפקודות והתגובות שמתקבלות מהמשתמש, ומבצע אינטראקציה עם רכיבים שונים על מנת להבטיח חוויית משתמש חלקה ואינטואיטיבית.

#### 1. רכיבי GUI Elements —> ניהול GUI Management

* **מטרה:** להכיל את כל האלמנטים הגרפיים כמו כפתורים, תפריטים ושטחי תצוגה.
* **אינטראקציה עם ניהול GUI:** רכיבי ה-GUI שולחים פקודות ונתונים לניהול ה-GUI.
  + **תגובה של ניהול GUI:** מנהל את ההצגה של כל האלמנטים, מעדכן את התצוגה על סמך פעולות המשתמש ומעביר את המידע לרכיב הלוגיקה.

#### 2. ניהול GUI —> רכיב הלוגיקה

* **מטרה:** לספק לרכיב הלוגיקה מידע על פעולות המשתמש ולבצע חיבור עם רכיבי המערכת.
* **אינטראקציה עם רכיב הלוגיקה:** ניהול ה-GUI מעביר פקודות ונתונים על פעולות המשתמש לרכיב הלוגיקה.
  + **תגובה של רכיב הלוגיקה:** מנתח את המידע המתקבל, מקבל החלטות על סמך הפקודות ומבצע את הפעולות הנדרשות.

#### 3. רכיב הלוגיקה —> תקשורת עם Sensor Nodes

* **מטרה:** לבצע את התקשורת עם הצמתים הסנסוריים על סמך ההוראות שהתקבלו מהמשתמש.
* **אינטראקציה עם Sensor Nodes:** רכיב הלוגיקה שולח פקודות או מקבל נתונים מהצמתים הסנסוריים.
  + **תגובה של Sensor Nodes:** מחזירים נתונים או מאשרים את הפעולות שנעשו על סמך הפקודות שהתקבלו.

### דוגמת תהליך בתוך רכיב הלוגיקה של Client GUI

1. **הפעולה של המשתמש:** המשתמש לוחץ על כפתור או מבצע פעולה אחרת על רכיב GUI **—>** ניהול GUI מעביר את הפקודה לרכיב הלוגיקה.
2. **קבלת הפקודה:** רכיב הלוגיקה מקבל את הפקודה ומבצע ניתוח **—>** הוא מחליט על פעולות נדרשות (כמו שליחת נתונים לצמתים הסנסוריים).
3. **תקשורת עם Sensor Nodes:** רכיב הלוגיקה שולח פקודות לצמתים הסנסוריים **—>** הצמתים מחזירים נתונים או אישורים.

### Sensor Nodes - רכיב הלוגיקה ותפקידו

**תפקיד רכיב הלוגיקה:**רכיב הלוגיקה בתוך Sensor Nodes אחראי על ניהול התקשורת עם ה-Sink, טיפול בנתונים המתקבלים והעברת מידע, כמו גם על ניהול כישלונות ועיבוד בצל. רכיב זה מרכז את כל ההגדרות וההגיון הנדרשים על מנת להבטיח שהצמתים הסנסוריים פועלים בצורה תקינה ויעילה.

#### 1. תקשורת עם Sink —> רכיב הלוגיקה

* **מטרה:** לקבוע את הנתיב להעברת הנתונים ל-Sink ולקבל הוראות ממנו.
* **אינטראקציה עם רכיב הלוגיקה:** רכיב התקשורת שולח נתונים והודעות מ-Sensor Nodes לרכיב הלוגיקה.
  + **תגובה של רכיב הלוגיקה:** מנתח את המידע המתקבל, מקבל החלטות לגבי ניהול הנתיב או העברת הנתונים ל-Sink.

#### 2. רכיב הלוגיקה —> ניהול כישלונות (Failure Management)

* **מטרה:** להתמודד עם תקלות או בעיות תקשורת שעלולות להתרחש במהלך הפעולה.
* **אינטראקציה עם ניהול כישלונות:** רכיב הלוגיקה מזהה בעיות ומעביר נתונים לניהול הכישלונות.
  + **תגובה של ניהול כישלונות:** מבצע צעדים לתיקון הבעיה, כמו ניסיונות חוזרים לשלוח נתונים, שינוי נתיב או הפניית הנתונים לצומת אחר.

#### 3. רכיב הלוגיקה —> עיבוד (Onion Processing)

* **מטרה:** לבצע את העיבוד הנדרש של הנתונים המוצפנים לפני שליחתם ל-Sink.
* **אינטראקציה עם עיבוד בצל:** רכיב הלוגיקה שולח נתונים לצורך עיבוד והצפנה.
  + **תגובה של עיבוד בצל:** מפענח את הנתונים, מפעיל שכבות הצפנה, ומכין את הנתונים למשלוח ל-Sink.

### דוגמת תהליך בתוך רכיב הלוגיקה של Sensor Nodes

1. **קבלת נתונים:** Sensor Nodes מקבלים נתונים מה-Sink **—>** רכיב הלוגיקה מנתח את המידע ומחליט על הצעדים הבאים.
2. **ניהול כישלונות:** אם מתרחשת בעיה בהעברת הנתונים, רכיב הלוגיקה מזהה את הבעיה ומעביר נתונים לניהול הכישלונות **—>** ניהול הכישלונות פועל לתקן את הבעיה.
3. **עיבוד בצל:** רכיב הלוגיקה שולח את הנתונים לעיבוד בצל **—>** עיבוד בצל מפעיל את התהליך הנדרש ומחזיר את התוצאה לרכיב הלוגיקה.

| **Task Description** | **Components Interaction** |
| --- | --- |
| Saving Keys | OnionManager (Logic > Layer Management > Communication with Libsodium) |
| Node Validation | OnionValidator (Failure Detection > Retry Mechanism > Logging) |
| Node Connection | SensorNodes (Communication with Sink > Logic > Communication with Client GUI) |
| Data Transfer Through Nodes | SensorNodes (Onion Processing > Logic > Communication with Sink) |
| Route Selection for Data Transmission | Sink (Logic > Route Management > Communication with SensorNodes) |
| Final Node in the Path (Decrypting Data) | SensorNodes (Onion Processing > Logic > Failure Management) |
| TCP Protocol Implementation | WsnConstructor (Network Topology Setup > Protocol Configuration > Logic) |
| Logging and Performance Data | OutputManager (Data Logging > Logic > File and Console Output) |
| Simulation Setup | WsnConstructor (Network Topology Setup > Protocol Configuration > Resource Management) |
| Post-Simulation Analysis | OutputManager (Post-Simulation Analysis > Logic > File and Console Output) |
| **שכבות ההצפנה** | **רכיב ( לוגיק > Layer Management )** |
| **הצפנת AES** | **רכיב OnionManager ( לוגיק > Communication with Libsodium < Layer Management )** |

דרך טובה להבין את המערכת היא דרך הנתונים ויישויות המידע שעוברים ונשמרים בה.

הכוונה היא **למידע שמועבר בתוך המערכת** (למשל בין מודולים / רכיבים שונים, או בין לקוח לשרת על גבי הרשת) וגם **למידע שנשמר על ידי המערכת** (למשל ב-DB, קבצי טקסט או בינאריים). שימו לב כי יישויות יכולות להכיל יישויות אחרות:

| 1. מידע מסוג … עובר בין … ל …. / מידע מסוג … נשמר ב …. 2. מידע מסוג <std::vector<unsigned char עובר בין הצמתים להעביר את המידע המוצפן   **מידע שנשמר על ידי המערכת -**   1. מידע מסוג json שנשמר בשרת דירקטורי - הקובץ מכיל מידע על המפתחות הפומביים 2. טבלאות נתונים ב -DB -  * טבלה של נתוני משתמשים (שם משתמש, סיסמאות ומזהה) * טבלת נתונים על המפתחות הפומביים   **🗑️** שאלו את עצמכם: מה סוג המידע שעובר במערכת? האם הוא נשמר או לא? אם הוא נשמר, היכן הוא נשמר? בין אילו רכיבים הוא עובר?  לדוגמה:  המידע המועבר בתוך המערכת שלנו הוא בקשות (לא נשמרות) ומידע (נשמר):   1. **בקשות ומענות בין השרת לרכיב הסריקות:**    1. בקשת אישור הרשמה / התחברות    2. מענה על אישור הרשמה / התחברות    3. בקשת אישור הרשאה של משתמש לרכיב    4. מענה על אישור הרשאה של משתמש לרכיב    5. בקשת שליחת קובץ LOG למייל.    6. מענה על שליחת קובץ LOG למייל. 2. **בקשות ומענות בין הלקוח לרכיב:**    1. בקשת התחברות / הרשמה    2. מענה על בקשת התחברות / הרשמה    3. בקשת הפעלת סריקה.    4. קבלת נתוני הסריקה.    5. בקשת שליחת הסריקה למייל. 3. **מידע העובר במערכת:**    1. נתוני משתמשים (שם משתמש, סיסמא וכו')    2. תוצאות סריקות **🗑️** |
| --- |



1. 
2. **פרק 4: תוכנית עבודה**

**🗑️ לקרוא ואז למחוק**

**🤩**לאחר שמסמך הארכיטקטורה של [שם הפרויקט] הושלם ואושר, יש לפרוט אותו לשלבים שונים על גבי ציר הזמן.

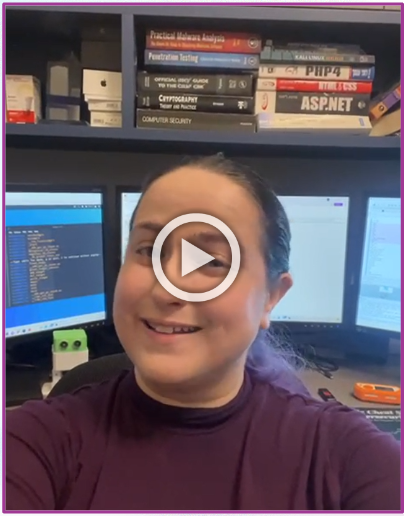
**למה ליצור תוכנית עבודה?**

מצפים לכם כמה חודשים של פיתוח, ולכן יש לתכנן מראש את סדר שלבי הפיתוח בצורה חכמה והולמת. לא כל הדרישות נולדו שוות, וגם לא תוכלו לפתח את כולן בבת-אחת. לכן יש להחליט **מהי העדיפות של כל דרישה**, ובהתאם לכך מה יהיה סדר הפיתוח שלהן.

בעת מתן עדיפות לדרישות, כלומר איזו דרישה תפותח קודם, השיקולים צריכים להיות בראש ובראשונה של תוכן - ולא של אילוצים טכניים. כלומר יש להתחיל מהרכיבים והדרישות שמהווים את עיקרי המהות של הפרויקט (הליבה הטכנולוגית), ולאחר מכן לבנות סביבם את יתר המערכת.

זוהי גישה של **מוצר מינימלי מתפקד** - כך שבסיומה של כל איטרציה ניתן יהיה להדגים פרויקט רץ ומתפקד (גם אם באופן מאוד חלקי ולא הכי יציב או מתוחכם). כמובן שגם כל איטרציה אמורה להציג פרויקט רץ אשר מתפקד יותר טוב או עם יותר פיצ'רים מאשר זה שהוצג באיטרציה (ספרינט) הקודמת.

**רוצים לשמוע את טיפ הזהב של נופר? 🔥**

  
**נופר, מפתחת תוכנה וחובבת אתגרי CTF**

* 1. **משימות ותלויות**

את סעיף זה יש למלא על בסיס פירוט הדרישות שכתבתם בפרק האפיון, ובהתאם לרכיבים של מסמך הארכיטקטורה.

מכיוון שכל דרישה מורכבת ממספר שלבים (שכל אחד מהם הוא פיצ'ר בפני עצמו), יש להתייחס לכל שלב/פיצ'ר כזה בהתאם ובנפרד מהיתר. בשלב הראשון תמספרו את הפיצ'רים שלכם ולאחר מכן תפרטו אותן למשימות בטבלה. שימו לב שמשימה צריכה להיות תחת רכיב אחד. במידה והיא נמצאת תחת מספר רכיבים, יש לפרוט את המשימה עוד יותר. בנוסף, חשבו היטב על **סדר המשימות** ברצף הפיתוח. ספציפית, האם יש משימות שתלויות בהשלמה של משימות אחרות קודם לכן.

1. פיצ'ר
2. פיצ'ר
3. פיצ'ר
4. פיצ'ר
5. פיצ'ר
6. פיצ'ר
7. פיצ'ר
8. פיצ'ר
9. פיצ'ר
10. פיצ'ר

| **מס' פיצ'ר** | **האם המשימה תלויה במשימה אחרת?** | **משימה** | **הערות למשימה** | **רכיב רלוונטי** |
| --- | --- | --- | --- | --- |
| TCP | **x** | להכין את הפרוטוקול ב C++ | להתייעץ עם אנשים נוספים אשר חקרו את הנושא ו-או יש להם ניסוי במימושו. |  |
| שכבות הצפנה | הצפנת AES | להכין את הפונקציה  בהתאם לאלגוריתם |  | OnionManager (Encryption Layer) |
| הצפנת AES | **x** | להכין את אלגוריתם ההצפנה בהתאם למחקר שעשינו |  | OnionManager (Encryption Layer) |
| הצמתים וחיבורם לרשת | מימוש שרת דיירקטורי | ביסוס תקשורת בין הצמתים לשרת |  | SensorNodes (Client Nodes) |
| הצומת האחרון במסלול | **x** | לוודא כי הצומת האחרונה במסלול שנקבע מפענחת את ההצפנה, ומעבירה את החבילה ליעד הרצוי |  | Communication with client GUI |
| בחירת המסלול דרכו תעבור החבילה | דוקרים | יצירת פונקציה אחראית ליצירת את המסלול בצורה רנדומלית |  | SensorNodes (Client Nodes) |
| העברת המידע דרך הצמתים | TCP | כתיבת פרוטוקול רשת |  | SensorNodes (Client Nodes) |
| ניהול הצמתים | הכנת הצומת (node) | ביסוס שרת דיירקטורי המנטר את הצמתים |  | Sink (Directory Server) |
| העברת קבצים ברשת TOR | חלוקת לסגמנטים בTCP ויצירת ההצפנות | הכנת הפונקציות  של TCP לחלוקה לסגמנטים |  | SensorNodes (Client Nodes) |
| הצפנה א-סימטרית עם מפתחות סימטריים | הצפנת AES והצפנת RSA |  |  | OnionManager (Encryption Layer) |
| שמירת מפתחות פומביים של הצמתים | הכנת בסיס נתונים וטבלאות רלוונטיות | פונקציות אשר אחראיות ליצירת המפתחות הפומביים, אחסונם במקום המתאים (קובץ בשרת דיירקטורי) |  | Sink (Directory Server) |
| התחברות למערכת |  | יצירת מסד נתונים אשר מכיל את הלקוחות |  | Sink (Directory Server) |
| שימוש בDOCKER ברשת TOR | **x** | כתיבת קוד לצמתים ברשת TOR ויצירת DOCKERfile |  |  |
| העברת סרטונים ברשת TOR | העברת קבצים ברשת TOR, יצירת שרתים | יצירה של אלגורתם  אשר מחלק את הקובץ לסיגמנטים |  | client GUI \ SensorNodes (Client Nodes) |

| רשימת הפיצ'רים:  רשימת פי   1. הצגת רשתות WiFi זמינות והתחברות לאחת מהן 2. לכידת התעבורה שעוברת ברשת 3. זיהוי התקפות והגנה מפניהן 4. זיהוי התקפת Man In The Middle 5. זיהוי התקפת Evil twin  | 1 | הצגת רשתות WiFi זמינות למשתמש | הפיצ'ר תלוי בכך שהכרטיס רשת תומך במצב MONITOR ואף הועבר למצב זה | את הרשתות ניקח באמצעות סריקת BEACON-ים | רכיב האותנטיקציה | | --- | --- | --- | --- | --- | | 2 | התחברות לרשת אותה בחר המשתמש | הפיצ'ר תלוי בכך שהוצגה רשימת רשתות למשתמש ובחירתו התקבלה | המשתמש יבחר את הרשת הרצויה בפלטפורמת WEB | רכיב האותנטיקציה |   **🗑️** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

* 1. **איטרציות (ספרינטים)**

זהו החלק האחרון של מסמך תכנות העבודה, ותבצעו אותו רק לאחר שקיבלתם לכך אישור מראש הצוות שלכם.

כמובן שיש להעזר במנטור במידה ויש לכם אחד, על מנת לתכנן נכון את אופן העבודה.

יש לזכור כי כל איטרציה דורשת חלוקת אחריות בין כל חברי הצוות - היא מתחילה בתכנון מפורט של הרכיבים המפותחים (בהתאם למסמכים שכתבתם עד כה) ומסתיימת בהצגת פרויקט רץ.

עליכם **לתעדף את המשימות** לפי סדר פיתוח. **חלקו אותם לאיטרציות שונות** על פי שיקולים של תלויות וסדר חשיבות.

| 1. איטרציה 1:    1. התאמת הפקטה ( בין לקוח לקוח ) לצורכי הרשת ובנייתה בהתאם נמצא בפיצרים ( העברת המידע דרך הצמתים , הצומת האחרון במסלול )    2. יצירת מעגל בקשה-תשובה בין הלקוח לשרת    3. חיבור השרת לצומת (נתב ההודעות המועברות בין הלקוח לשרת) בפיצר ( הצמתים וחיבורם לרשת)    4. הרחבת בניית בקשות handel request    5. handel new clinet - פיתוח מענה בקשות 2. איטרציה 2: |
| --- |



2. 
3. **פרק 5: עיצוב**

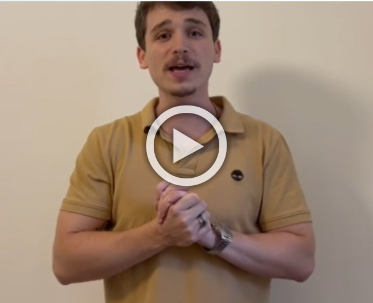
**🗑️ לקרוא ואז למחוק**

אתם כבר כמעט בסוף תהליך הכתיבה…זהו השלב האחרון במסמך הפרויקט שלכם - אותו תרחיבו בתחילת כל איטרציה.

התוכן של פרק זה עשוי להיות שונה עבור כל איטרציה, בהתאם לדגשים ולנושאים המוכלים בה. יש לכלול תכנון מפורט עבור כל איטרציה, בצירוף שרטוטים והסברים.

**מה מטרת הפרק?**

לאחר שכבר החלטתם אילו פיצ'רים מפותחים בכל איטרציה, הגיע הזמן להוריד לקרקע את התוכניות והמשימות שהגדרתם לכל איטרציה ולעצב את תוכנית המימוש שלה בפועל.   
  
מניסיון שלנו, בלי לוחות זמנים והגדרת מטרות קל מאוד לאבד תחושת זמן וללכת לאיבוד בים המשימות. בפרק זה תגדירו מטרות מוגדרות לכל איטרציה ותיצרו לוחות זמנים וחלוקת עבודה צוותית. בקיצור, **אתם מרחק נגיעה מ"ללכלך את הידיים"!** איזה כיף!

**המקרה שקרה לעומרי בצבא כשדילג על כתיבת מסמך עיצוב…**  
👆 **לחצו כדי לשמוע את הסיפור של עמרי,  
בוגר 8200 וחוקר חולשות בסטארטאפ**

* 1. **👂 שימו לב: יש 5 איטרציות ועבור כל אחת מהן תצטרכו לשכפל את החלק שמופיע למטה ולמלא אותו בהתאם.  
       
     איטרציה 1**

חזון:

| מטרות האיטרציה הן …  בסיום האיטרציה יהיה תוצר שעושה …  **🗑️** שאלו את עצמכם: מה אתם מצפים להשיג בסוף האיטרציה?  לדוגמה:   * בסיום האיטרציה הנ"ל, תוצג למשתמש רשימה של רשתות WIFI מהן יצטרך לבחור אחת. אם קיימת אבטחה, ידרש המשתמש להקיש סיסמא ולאחר מכן הרכיב יחובר לרשת באופן אוטומטי. * בסיום האיטרציה אנו מצפים שנצליח להסניף את כל תעבורת הרשת.**🗑️** |
| --- |

משימות עיקריות

| המשימות העיקריות של האיטרציה הזו:   1. משימה … שעושה …   **🗑️** שאלו את עצמכם: מה המשימות העיקריות שאתם מצפים להשלים ו/או להתקדם בהם עד סוף האיטרציה? על מה אתם שמים את הפוקוס בתעדוף? **הוסיפו צילום/תמונה אם צריך**.  לדוגמה:   * ההתחברות לרשת תעשה בעיקר על ידי פקודות מערכת (System calls) ולכן שלושת הימים הראשונים יוקדש הזמן למחקר על קריאות מערכת בנושא. את מציאת הרשתות הזמינות נממש באמצעות הסנפת חבילות ביקון (חבילות מידע שACCESS POINTים שולחים על מנת לידע על קיומם) לצורך כך יוקדש שבוע. לאחר מכן, נעבור למימוש בקוד בו נשלב את הפקודות השונות על מנת להגיע לחזון שהגדרנו לעצמנו. * מישור ההסנפות יתפוס חלק קטן יותר מהספרינט אך יהווה חלק משמעותי יותר בהמשך הפרוייקט ולכן עלינו להשקיע יותר במחקר. כמו כן, לצורך הסנפת הרשת יש צורך להעביר את כרטיס הרשת למצב monitor mode. למחקר על העבר הכרטיס למצב מוניטור והבנה של המשמעויות הכרוכות בזה יוקדש שבוע וחצי שבסופו אנו רוצים להצליח להעביר את כרטיס הרשת למצב זה ולדעת להשתמש בו. שנצליח, אנו רוצים גם לקנפג את Scapy כך שישתמש בכרטיס הרשת שהועבר למצב מוניטור ולהפיס את כל התעבורה ברשת לתוך קובץ. לצורך כך אנו מקדישים שלושה ימים.**🗑️** |
| --- |

שרטוט של רכיב שאתם הולכים לממש בספרינט:

| שרטוט (מעולם הנדסת התוכנה) של רכיב לבחירתכם שאתם הולכים לממש בספרינט:  קובץ **🗑️** |
| --- |

* 1. **אם הגעתם לכאן, מגיע לכם כל הכבוד…  
     סיימתם את הפרויקט שלכם, איזה יופי! 🎉🥳🙌**