מטלת מנחה (ממ״ן) 15

הקורס: 20937 - תכנות מערכות דפנסיבי

חומר הלימוד למטלה: יחידות 1-7

סמסטר: 2021א

חשוב!

קראו היטב את כל המטלה לפני תחילת העבודה. וודאו שאתם מבינים היטב את פרוטוקול התקשורת ואת המבנה של תוכנת השרת והלקוח.

ארכיטקטורה

ארכיטקטורת התוכנה שנפתח מבוססת על שרת-לקוח. כלומר, ההודעות נשלחות ראשית לשרת, ולאחר מכן, השרת מפנה לכל לקוח את ההודעה המתאימה.

תוכנת הלקוח ״תמשוך״ הודעות מהשרת באופן מחזורי (pull request).

מאפיין זה מאפשר לשלוח הודעות ללקוחות במצב מחובר (online) וגם במצב מנותק (offline).

בנוסף, נתמוך בהצפנה מקצה לקצה[1] (end-to-end encryption), כלומר, ההודעות מוצפנות בצד הלקוח ונשלחות בצורה מוצפנת, כך שאף אחד לא יכול לפענח את המידע למעט לקוח היעד (גם לא השרת).

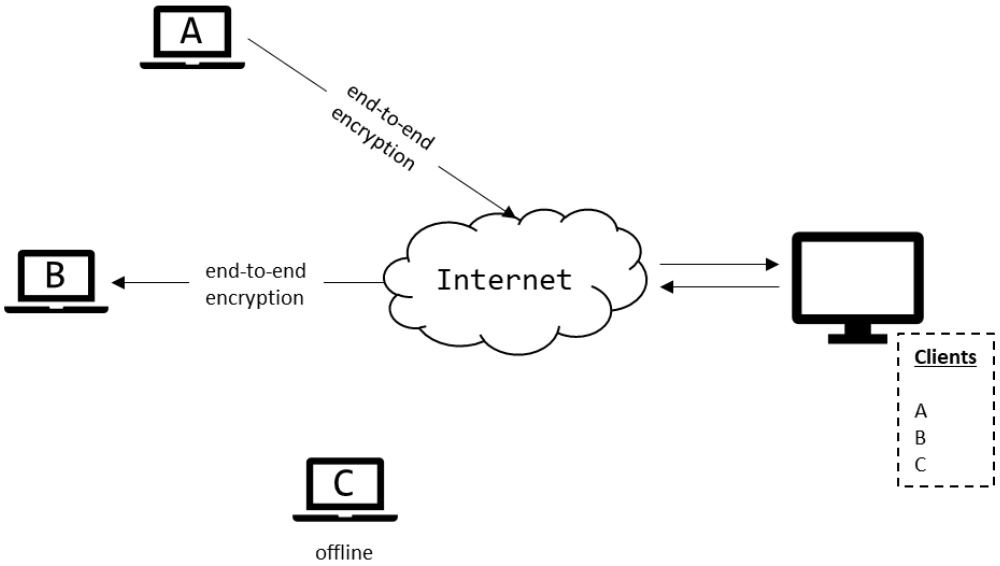

Figure 1 ארכיטקטורת שרת-לקוח שתומכת בהצפנה מקצה לקצה

[1] https://en.wikipedia.org/wiki/End-to-end_encryption

שרת

תפקיד השרת לנהל את רשימת המשתמשים הרשומים לשירות ולאפשר להם להחליף ביניהם הודעות מסוגים שונים.

א. השרת יכתב בשפת python
ב. השרת יתמוך בפרוטוקול חסר מצב (stateless)[2], כלומר, לא ישמור מידע בין בקשה לבקשה (כל בקשה עומדת בפני עצמה).
ג. השרת יתמוך בריבוי משתמשים ע"י תהליכונים (threads) או ע"י selector
ד. גירסת השרת תהיה **1**

פורט

השרת יקרא את מספר הפורט מתוך קובץ טקסט בצורה הבאה:

- שם הקובץ: port.info
- מיקום הקובץ: באותה תיקיה של קבצי הקוד של השרת
- תוכן הקובץ: מספר פורט
לדוגמא:
1234

נתונים

השרת ישמור את נתוני הלקוחות וההודעות בזיכרון (RAM).

מומלץ להגדיר מחלקה עבור רשומת לקוח ומחלקה עבור רשומת הודעה בהתאם למבנה הטבלאות המוגדר בסעיף "בונוס".

בונוס (10 נקודות)

שמירת הנתונים תעשה ע"י טבלאות SQL בקובץ בשם server.db

אם בחרתם לממש את הבונוס, גירסת השרת תהיה **2**.

שימו לב: בתחילת הריצה של השרת, אם הקובץ לא קיים יש ליצור אותו. באותו אופן, אם טבלה לא קיימת, יש ליצור אותה.

מידע על הלקוחות ישמר בטבלה בשם clients

מבנה הטבלה:

שם	סוג	הערות
ID	16 בתים (128 ביט)	מזהה ייחודי עבור כל לקוח. אינדקס
Name	מחרוזת (255 בתים)	מחרוזת ASCII המייצגת שם משתמש. **כולל תו מסיים!** (null terminated)
PublicKey	160 בתים	מפתח ציבורי של לקוח
LastSeen	תאריך ושעה	הזמן בו התקבלה בקשה אחרונה מלקוח

מידע על הודעות ישמר בטבלה בשם messages

[2] קראו כאן על פרוטוקול חסר מצב: https://en.wikipedia.org/wiki/Stateless_protocol

מבנה הטבלה:

שם	סוג	הערות
ID	4 בתים	אינדקס
ToClient	16 בתים	מזהה ייחודי של המקבל
FromClient	16 בתים	מזהה ייחודי של השולח
Type	בית	סוג ההודעה
Content	Blob	תוכן ההודעה

אופן פעולת השרת

1. קורא את הפורט מתוך הקובץ port.info
2. ממתין לבקשות מלקוחות בלולאה אין סופית
3. בעת קבלת בקשה מפענח את הבקשה בהתאם לפרוטוקול:

א. בקשה לרישום: במידה ושם המשתמש המבוקש כבר קיים, השרת יחזיר שגיאה. אחרת, השרת ייצר UUID חדש עובר המשתמש, ישמור את הנתונים בזיכרון (או בבסיס הנתונים) ויחזיר תשובת הצלחה

ב. בקשת לרשימת לקוחות: השרת יחזיר את רשימת הלקוחות לפי הפרוטוקול.

ג. בקשת שליחת הודעה תטופל באופן הבא:
השרת יחלץ את סוג ההודעה ואת תוכן ההודעה (מתוך ה- payload) וישמור אותו בזיכרון (או בבסיס הנתונים)

ד. בקשת שליפת הודעות ממתינות תטופל באופן הבא:
השרת ישלוף הודעות מתוך הזיכרון (או בסיס הנתונים) וישלח תשובה ללקוח
שימו לב! הודעות שנשלחו בהצלחה ללקוח ימחקו.

לקוח

תוכנת הלקוח תאפשר למשתמש לשלוח הודעה למשתמש אחר אשר רשום בשרת.

שימו לב! ניתן לשלוח הודעה לכל משתמש רשום במערכת (אין "אנשי קשר").

א. תוכנת הלקוח תכתב בשפת C++
ב. הלקוח ירוץ במצב מסוף (**console**) ויקבל קלט מהמשתמש (**stdin**) לביצוע הפעולות השונות
ג. הלקוח יתמוך בהצפנה מקצה לקצה
ד. גירסת הלקוח תהיה **1**

כתובת השרת והפורט

הלקוח יקרא את כתובת השרת והפורט מתוך קובץ טקסט בצורה הבאה:

- שם הקובץ: server.info
- מיקום הקובץ: בתיקיה של קובץ ההרצה (.exe)
- תוכן הקובץ: כתובת IP + נקודותיים + מספר פורט
  לדוגמא:
  127.0.0.1:1234

שם ומזהה ייחודי[3]

הלקוח ישמור ויקרא את השם והמזהה הייחודי שלו מתוך קובץ טקסט בצורה הבאה:

- שם הקובץ: me.info
- מיקום הקובץ: בתיקיה של קובץ ההרצה (.exe)
- תוכן הקובץ:
  שורה ראשונה: שם
  שורה שניה: מזהה ייחודי בייצוג ASCII כאשר כל שני תווים מייצגים ערך hex בעל 8 סיביות.
  שורה שלישית: מפתח פרטי שנוצר בריצה הראשונה של התוכנית בפורמט בסיס 64.
  לדוגמא:

```
Michael Jackson
64f3f63985f04beb81a0e43321880182
MIGdMA0GCSqGSIb3DQEBA …
```

נתונים

הלקוח ישמור את נתוני הלקוחות (מזהה ייחודי, שם, מפתח ציבורי ומפתח סימטרי) בזיכרון (RAM)

אופן פעולת הלקוח

הלקוח יציג למשתמש את התפריט כדלהלן וימתין לקלט מהמשתמש במצב מסוף (stdin) בלולאה אין סופית.

```
MessageU client at your service.
```

[3] בתרגיל זה נעשה שימוש במזהה ייחודי גלובלי (UUID). לקריאה נוספת:
https://en.wikipedia.org/wiki/Universally_unique_identifier

```
 1) Register
 2) Request for clients list
 3) Request for public key
 4) Request for waiting messages
 5) Send a text message
51) Send a request for symmetric key
52) Send your symmetric key
 0) Exit client
?
```

שגיאה מצד השרת

בכל מקרה של שגיאה הלקוח ידפיס למסך הודעה: "server responded with an error" ויחכה לקלט הבא.

פעולות אפשריות

בקשת רישום – קלט "1"

1. הלקוח יקלוט שם משתמש מהמסוף וישלח בקשת רישום לשרת.
2. הלקוח ישמור בקובץ בשם **me.info** את השם והמזהה הייחודי שיקבל מהשרת.

שימו לב! במידה והקובץ כבר קיים הלקוח לא יאפשר בקשת רישום וידפיס שגיאה למסוף.

בקשת רשימת לקוחות – קלט "2"

1. הלקוח ישלח בקשת רשימת לקוחות לשרת. יפענח את התשובה וידפיס למסך את שמות רשימת הלקוחות

בקשת מפתח ציבורי – קלט "3"

1. הלקוח ישלח בקשת מפתח ציבורי לשרת.

בקשת שליפת הודעות ממתינות – קלט "4"

1. הלקוח ישלח בקשת שליפת הודעות ממתינות לשרת. יפענח את התשובה וידפיס למסך את ההודעות בצורה הבאה:

```
From: <user name>
Content:
<content>
.
.
-----<EOM>-----
\n
```

א. עבור סוג הודעה "בקשת מפתח סימטרי" יש לכתוב בתוכן ההודעה: " Request for symmetric key"

ב. עבור סוג הודעה "שליחת מפתח סימטרי" יש לכתוב בתוכן ההודעה: " symmetric key received". בנוסף, תוכנת הלקוח תשמור בזיכרון את המפתח הסימטרי עבור לקוח זה.

ג. עבור סוג הודעה "שליחת הודעת טקסט" יש לפענח את המסר באמצעות המפתח הסימטרי ולהציג את ההודעה. אם לא קיים מפתח סימטרי או שאינו תקין, יש לכתוב "can't decrypt message"

בקשת שליחת הודעה – קלט "5"

1. הלקוח ימתין לקליטת שם משתמש **היעד** מהמסוף
2. הלקוח ימתין לקליטת הודעת טקסט
3. הלקוח ישלח בקשת "שליחת הודעה" מסוג "שליחת הודעת טקסט" לשרת.

בקשת קבלת מפתח סימטרי – קלט "51"

1. הלקוח ימתין לקליטת שם משתמש היעד מהמסוף
2. הלקוח ישלח בקשת "שליחת הודעה" מסוג מפתח סימטרי.

בקשת שליחת מפתח סימטרי – קלט "52"

1. הלקוח ימתין לקליטת שם משתמש היעד מהמסוף
2. הלקוח ייצר מפתח סימטרי. ישמור אותו בזיכרון עבור לקוח היעד וישלח בקשת "שליחת הודעה" מסוג שליחת מפתח סימטרי.

בונוס (10 נקודות)

בקשת שליחת קובץ – קלט "50"

1. הלקוח ימתין לקליטת שם משתמש **היעד** מהמסוף
2. הלקוח ימתין לקליטת שם קובץ (כולל הנתיב המלא)
   - אין להוסיף תמיכה ב- Unicode (כלומר, רק תווי ascii)
   - במידה והקובץ לא נמצא או שהנתיב המוזן לא תקין, יש להדפיס שגיאה "file not found"
3. הלקוח ישלח בקשת "שליחת הודעה" מסוג "שליחת קובץ" לשרת.

שימו לב!

- יש להוסיף לתפריט את האפשרות: `50) Send a file`
- בעת <u>קבלת הודעת</u> "שליחת קובץ" יש לפענח את המסר באמצעות המפתח הסימטרי, לשמור את הקובץ בשם זמני בתיקיית %TMP% ולהציג את הנתיב המלא לקובץ במקום Content.
- גירסת הלקוח תהיה **2**

יציאה – קלט תו "0"

1. הלקוח ישחרר את משאבי מערכת ויסיים את ריצתו

פרוטוקול התקשורת

!	הפרוטוקול המתואר הוא מימוש בסיסי של הצפנה מקצה לקצה (end-to-end encryption). במידה ותרצו לממש מערכת תוכנה עם תמיכה בפרוטוקול דומה, **מומלץ** להעמיק בנושא.

כללי

- הפרוטוקול הוא בינארי וממומש מעל TCP.
- כל השדות המספריים חייבים להיות עם ערכים גדולים מאפס (**unsigned**) ומיוצגים כ- **little endian**
- פרוטוקול זה תומך **בבקשות** לשרת **ותשובות** ללקוח. בקשות או תשובות יכולות להכיל "**הודעה**".
- הודעה עוברת בין לקוחות

זכרו! הפרוטוקול מחייב ולא ניתן לעשות בו שינויים. כפועל יוצא, כל שרת ולקוח הממשים את הפרוטוקול יכולים לעבוד אחד מול השני.

רישום למערכת

1. כל לקוח שמתחבר בפעם הראשונה נרשם בשירות עם שם (מחרוזת באורך מקסימלי של 255 בתים) ומעביר את המפתח הציבורי שלו
2. השרת יחזיר ללקוח מזהה ייחודי שנוצר עבורו או שגיאה אם השם כבר קיים בבסיס הנתונים.

בקשות מהשרת

1. לקוח יכול לבקש את רשימת המשתמשים האחרים
2. לקוח יכול לבקש מפתח ציבורי של לקוח מסויים
3. לקוח יכול לבקש את כל ההודעות המחכות לו
4. לקוח יכול לשלוח הודעה ללקוח אחר

החלפת הודעות

1. לקוח **A** מבקש מהשרת את המפתח הציבורי של לקוח **B**
2. לקוח **A** שולח הודעה (דרך השרת) ללקוח **B** מסוג "בקשת מפתח הצפנה **סימטרי**"
   ההודעה מוצפנת ע"י המפתח **הציבורי** של **B**
3. השרת מקבל את ההודעה ושומר אותה
4. לקוח **B** מושך מהשרת את ההודעות הממתינות לו
5. לקוח **B** מפענח את ההודעה באמצעות המפתח **הפרטי**
6. לקוח **B** מבקש מהשרת את המפתח **הציבורי** של לקוח **A**
7. לקוח **B** שולח תשובה מסוג "מפתח הצפנה **סימטרי**" ללקוח **A**
   התשובה מוצפנת ע"י המפתח **הציבורי** של **A**
8. השרת מקבל את ההודעה ושומר אותה
9. לקוח **A** מושך מהשרת את ההודעות הממתינות לו
10. לקוח **A** מפענח את ההודעה באמצעות המפתח **הפרטי**
11. כעת, לקוח **A** ולקוח **B** יכולים לשוחח באמצעות מפתח הצפנה **סימטרי**

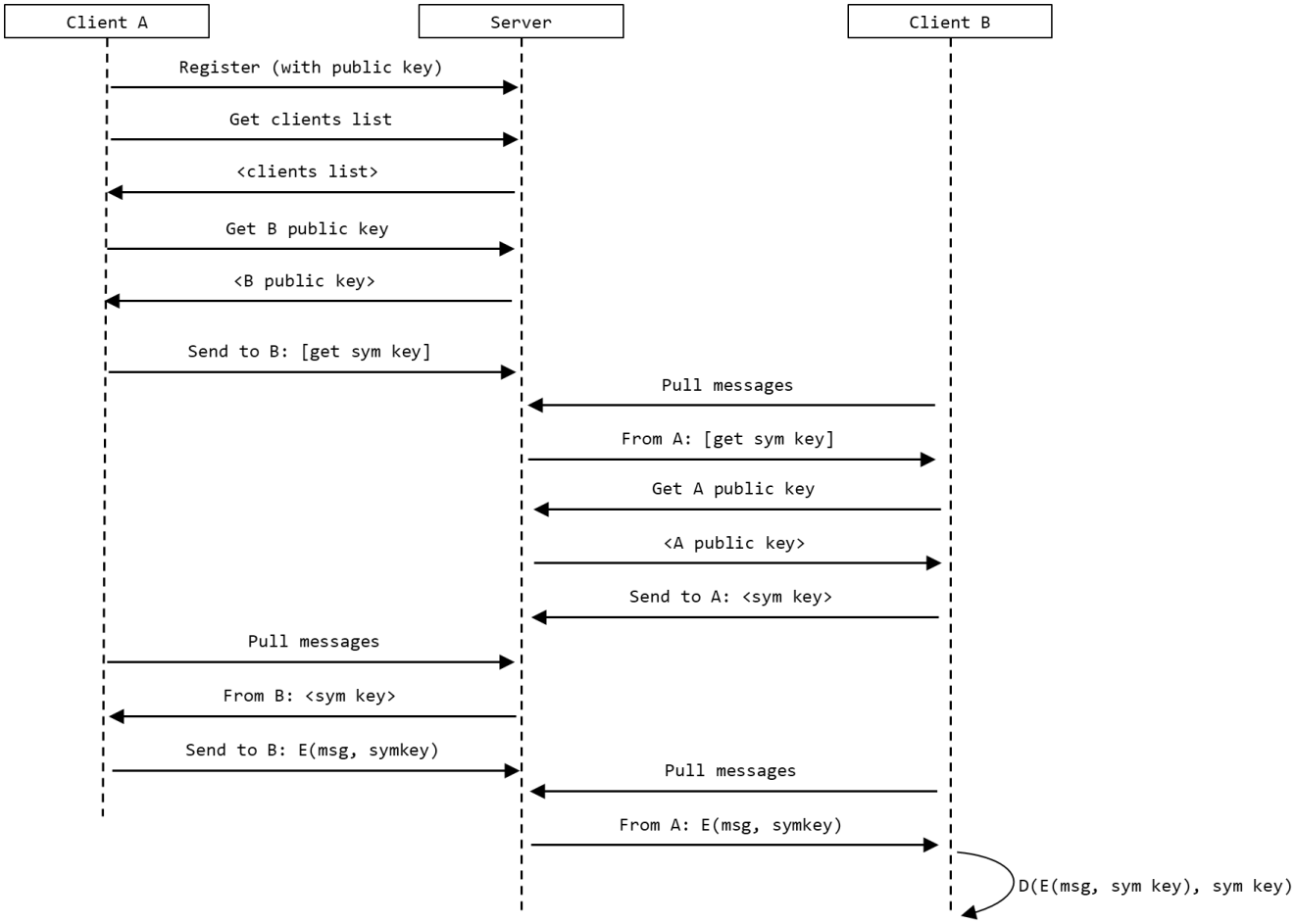

Figure 2 החלפת הודעות בין לקוח A ללקוח B

פרטי הפרוטוקול

בקשות

מבנה בקשה מהלקוח לשרת. השרת יפענח את התוכן (payload) לפי קוד הבקשה.

בקשה לשרת

Request	שדה	גודל	משמעות
כותרת (Header)	Client ID	16 בתים (128 ביט)	מזהה ייחודי עבור כל לקוח
	Version	בית	מספר גירסת לקוח
	Code	בית	קוד בקשה
	Payload size	4 בתים	גודל תוכן הבקשה
תוכן (payload)	payload	משתנה	תוכן הבקשה. משתנה בהתאם לבקשה

תוכן (payload)

התוכן משתנה בהתאם לבקשה. לכל בקשה מבנה שונה.

קוד בקשה 100 – רישום

שדה	גודל	משמעות
Name	255 בתים	מחרוזת ASCII המייצגת שם משתמש. **כולל תו מסיים!** (null terminated)
Public Key	160 בתים	מפתח ציבורי של לקוח

* **שימו לב:** השרת יתעלם מהשדה Client ID

קוד בקשה 101 – רשימת משתמשים

שדה payload לא קיים. שדה Payload size=0

קוד בקשה 102 – שליפת מפתח ציבורי של לקוח

שדה	גודל	משמעות
Client ID	16 בתים	מזהה ייחודי של לקוח

קוד בקשה 103 – שליחת הודעה ללקוח

שדה	גודל	משמעות
Client ID	16 בתים	מזהה ייחודי של לקוח **היעד**
Message Type	בית	סוג ההודעה ללקוח
Content Size	4 בתים	גודל תוכן ההודעה
Message Content	משתנה	תוכן ההודעה. **מוצפן** ע״י המפתח הציבורי של לקוח היעד או ע״י מפתח סימטרי. תלוי בסוג ההודעה.

* סוגי ההודעות מפורטים בהמשך

קוד בקשה 104 – שליפת הודעות ממתינות

שדה payload לא קיים. שדה Payload size=0

סוג ההודעה ללקוח (Message Type)

לקוח יכול לשלוח הודעות שונות ללקוח אחר.

סוג ההודעה 1 – בקשת מפתח סימטרי

תוכן ההודעה ריק. שדה Content Size=0

סוג ההודעה 2 – שליחת מפתח סימטרי

שדה Message Content מכיל מפתח סימטרי **מוצפן ע"י מפתח ציבורי** של לקוח היעד

סוג ההודעה 3 – שליחת הודעת טקסט

שדה Message Content מכיל טקסט **מוצפן ע"י מפתח סימטרי**.

סוג ההודעה 4 – שליחת קובץ

יש לממש רק אם בחרתם להוסיף את אפשרות שליחת הקובץ בצד הלקוח (סעיף בונוס).

שדה Message Content מכיל קובץ **מוצפן ע"י מפתח סימטרי**.

תשובות

תשובה מהשרת

Response	שדה	גודל	משמעות
כותרת (Header)	Version	בית	מספר גירסת שרת
	Code	2 בתים	קוד התשובה
	Payload size	4 בתים	גודל תוכן התשובה
תוכן (payload)	payload	משתנה	תוכן התשובה. משתנה בהתאם לתשובה

תוכן (payload)

קוד תשובה 1000 – רישום הצליח

שדה	גודל	משמעות

Client ID	16 בתים	מזהה ייחודי של לקוח

קוד תשובה 1001 – רשימת המשתמשים

שדה	גודל	משמעות
Client ID	16 בתים	מזהה ייחודי של לקוח
Client Name	255 בתים	מחרוזת ASCII המייצגת שם משתמש. **כולל תו מסיים!** (null terminated)

חשוב: רשימת משתמשים לא תכלול את המשתמש שביקש אותה. כמו כן, הרשימה עשויה לכלול משתמשים רבים. הם יופיעו אחד אחרי השני וניתן לחשב את מספרם ע"י הנוסחה:
Payload Size / (16+255)

קוד תשובה 1002 – מפתח ציבורי

שדה	גודל	משמעות
Client ID	16 בתים	מזהה ייחודי של לקוח
Public Key	160 בתים	מפתח ציבורי של לקוח

קוד תשובה 1003 – הודעה ללקוח נשלחה (שמורה אצל השרת – לא בהכרח נקראה)

שדה	גודל	משמעות
Client ID	16 בתים	מזהה ייחודי של לקוח **היעד**
Message ID	4 בתים	מזהה ייחודי של הודעה

קוד תשובה 1004 – שליפת הודעות ממתינות

שדה	גודל	משמעות
Client ID	16 בתים	מזהה ייחודי של הלקוח ממנו הגיעה ההודעה
Message ID	4 בתים	מזהה ייחודי של הודעה
Message Type	בית	סוג ההודעה ללקוח
Message Size	4 בתים	גודל ההודעה
Content	משתנה	תוכן ההודעה

חשוב: יכולות להיות הודעות ממתינות רבות. הן תופענה אחת אחרי השניה ברצף

קוד תשובה 9000 – שגיאה כללית

שדה payload לא קיים. שדה Payload size=0

הצפנה

פרוטוקול התקשורת משתמש בהצפנה סימטרית על מנת לקודד את ההודעה בין הלקוחות ובהצפנה אסימטרית על מנת להחליף מפתח בין הלקוחות.

בתרגיל זה השתמשו בספריה **Crypto++**[4]

הצפנה סימטרית

עבור הצפנה סימטרית השתמשו ב- AES-CBC.

אורך המפתח **128 ביט**. ניתן להניח שה- IV מאופס תמיד (הזיכרון מלא באפסים).

שימוש כזה ב- IV לא בטוח אם משתמשים באותו מפתח בכל פעם, אך לצורך הממן הוא מספק.

הצפנה אסימטרית

עבור הצפנה אסימטרית השתמשו ב- RSA.

אורך המפתחות **1024 ביט**.

שימו לב: הספריה Crypto++ מחזיקה מפתחות ציבוריים בפורמט X509[5]. פורמט זה מכיל Header לפני המפתח עצמו וערכים נוספים. לכן, גודלו הסופי (בצורה בינארית) הוא **160 בתים** (עבור מפתחות בגודל שונה גודלו הסופי של המפתח ישתנה בהתאם).

דגשים לפיתוח

1. מומלץ לעבוד עם מערכת לניהול קוד (כדוגמת גיט[6])
2. עבדו באופן מודולרי ובדקו את עצמכם כל הזמן
   א. זהו את המחלקות והפונקציות החשובות
   ב. **בצד השרת:**
   כיתבו קוד לטיפול בבקשה אחת. הוסיפו תמיכה בריבוי לקוחות בשלב מאוחר יותר
   ג. **בצד הלקוח:**
   ממשו את הרכיבים הגדולים באופן בלתי תלוי בחלקים אחרים של המערכת (תקשורת, הצפנה, פרוטוקול וכו').
3. ממשו קוד לבדיקה כבר בשלבים מוקדמים של הפרוייקט
   א. **בצד השרת:**
   השתמשו בהדפסות למסך או בכתיבה ללוג כדי לעקוב אחרי התקשורת. תוכלו גם לטעון את המודול לתוך ה- interpreter ולעבוד באופן דינמי.
   ב. **בצד הלקוח:**
   כיתבו פונקציות קטנות שבודקות חלקים נפרדים של המערכת. השתמשו בפונקציות הללו תוך כדי כתיבת הקוד עצמו.
4. כתיבת הקוד
   א. ממשו את התוכנה לפי עקרונות תכנות מונחה עצמים
   ב. שימו לב לייצוג ערכים בזיכרון כ- little-endian או big-endian
   ג. הקפידו על תיעוד של הקוד (comments)
   ד. תנו שמות משמעותיים למשתנים, פונקציות ומחלקות. המנעו ממספרי קסם!

---

[4] https://www.cryptopp.com/
[5] https://en.wikipedia.org/wiki/X.509
[6] https://www.atlassian.com/git/tutorials/what-is-version-control

ה. הודעה יכולה להיות גדולה מאוד (בגודל דינמי). חשבו על הדרך הנכונה ביותר לקבל ולשלוח כמות מידע גדולה.
ו. **אבטחת מידע** – חשבו לאורך כל הדרך על כתיבת קוד בטוח לפי העקרונות שלמדתם: האם בדקתם את הקלט? איך נעשה שימוש בזיכרון דינמי? האם מתבצעת המרת טיפוסים (casting) וכו'..

5. **לפני ההגשה**

א. בדקו שהפרוייקט מתקמפל ורץ בצורה תקינה ללא קריסות או תלויות בספריות שונות (למעט הספריות הנדרשות לתרגיל)
ב. מומלץ לייצר תיקיה חדשה ולהעתיק לשם את הקבצים המיועדים לשליחה. לייצר פרוייקט VS חדש, לקמפל ולהריץ
ג. העבודה תבדק על מ"ה חלונות עם Visual Studio Community 2019

דגשים לקוד שרת:

1. השתמשו בפייתון **גירסה 3**
2. עשו שימוש בספריות פייתון הסטנדרטיות
3. תוכלו להעזר בספריה **struct** על מנת לעבוד עם נתוני התקשורת בנוחות

דגשים לקוד לקוח:

1. מומלץ (אבל לא חובה) לעשות שימוש בספריות STL
2. ניתן ורצוי להשתמש ביכולות C++11 (לדוגמא פונקציות מסוג למדה, שימוש ב- auto וכו'..).
3. למימוש התקשורת עשו שימוש ב- winsock או בספרית boost

הגשה

שרת

1. עליכם להגיש רק את קבצי הקוד (כלומר קבצי py.).
שימו לב! על התוכנית להתקמפל ולרוץ בצורה תקינה (ללא צורך בתוספות קבצים ללא קריסות)
2. יש לכלול פונקציה ראשית בשם **main**. פונקציה זו תהיה הפונקציה הראשית של תוכנית השרת והיא תעבוד לפי אופן פעולת השרת המפורט לעיל.
טיפ:
תוכלו להשתמש במנגנון הבא כדי לאפשר עבודה אינטראקטיבית **וגם** הרצה של הקוד

```
if __name__ == "__main__":
```

לקוח

1. עליכם להגיש רק את קבצי הקוד (כלומר קבצי h. ו- cpp.).
שימו לב! על התוכנית לרוץ בצורה תקינה (ללא צורך בתוספות קבצים, ללא קריסות)
2. עבודתכם תיבדק במערכת הפעלה חלונות, באמצעות Visual Studio ולכן מומלץ לעבוד עם סביבה זו.