תרגיל בית 3

תיאור התרגיל

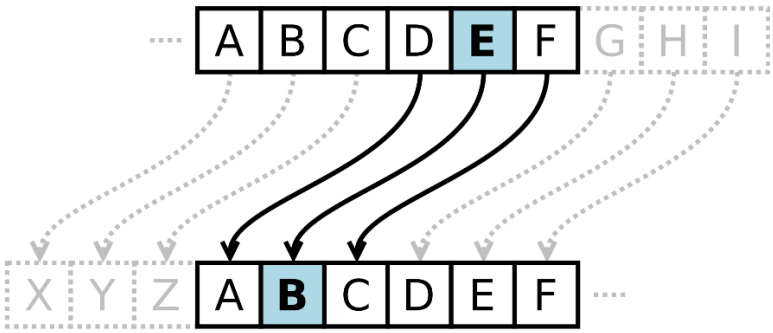
בתרגיל זה נממש מנהל התקן (Device Driver) עבור התקן הצפנה (Encryption Device) באמצעות שימוש ב- Kernel Module.

בעולם האמיתי, ייתכן והתקן ההצפנה, עימו מנהל ההתקן מתקשר, היה רכיב חומרתי אשר מאפשר לבצע פעולות הצפנה מורכבות במהירות. עם זאת, למטרת התרגיל, אנו נממש פעולות הצפנה פשוטות כחלק מהקוד של מנהל ההתקן, ורק נדמה את ההתקן החומרתי.

אם כן, בתרגיל אנו נדמה שני התקני הצפנה שונים:

הצפנת קיסר (Caesar Cipher)

בהינתן מחרוזת קלט `s` ומפתח הצפנה `k`, התקן ההצפנה מזיז (Shift) את ערכו של כל תו במחרוזת `s`, אל ערך הנמצא `k` מקומות אחריו (בהתאם לקידוד של ה- ASCII Table).



מידע נוסף ניתן למצוא כאן.

הצפנה

בהינתן מחרוזת קלט `s` ומפתח הצפנה `k`, פעולת ההצפנה מבוצעת באופן הבא: (קוד להמחשה בלבד)

```
for(i = 0; i < strlen(s); i++)

{

      s[i] = (s[i] + k) % 128;

}
```

פענוח

בהינתן מחרוזת קלט s ומפתח הצפנה k, פעולת הפענוח מבוצעת באופן הבא: (קוד להמחשה בלבד)

```
for(i = 0; i < strlen(s); i++)

{

      s[i] = ((s[i] - key) + 128) % 128;

}
```

הסבר

- השימוש באופרטור מודולו (%) הוא כדי שפעולת ההזזה (Shifting) תהיה פעולה מעגלית.
- השימוש במודולו 128 הינו משום שה- Character Encoding של המחרוזות בהן נעשה שימוש הוא מסוג ASCII. כלומר, ישנם רק 127 תווים שונים בהם ה- Shell יבחין כאשר נרצה להדפיס את המחרוזת למסך. לשימושכם, פירוט על ה- ASCII Table נמצא בקישור הבא.
- **הסבר:**

אם s[i] - key חיובי, אז קיבלנו את הערך התו המקורי. במקרה זה, פעולת השארית 128 מבטלת את התוספת של 128, ולכן עדיין מקבלים את ערך התו הנכון.
אם s[i] - key שלילי, אז זה אומר שערך התו המקורי נמצא בתא (s[i] - key) + 128 בטבלת ה- ASCII, לכן, הפעלת שארית 128 על ערך זה, שכבר נמצא בטווח שבין 0 ל- 127, עדיין תחזיר את הערך הנכון.

הצפנת XOR (XOR Cipher)

בהינתן מחרוזת קלט s ומפתח הצפנה k, התקן ההצפנה מחליף את ערכו של כל תו במחרוזת s, עם תוצאת חיבור XOR של ערך התו ומפתח ההצפה.

מידע נוסף ניתן למצוא כאן.

הצפנה ופענוח

בהינתן מחרוזת קלט s ומפתח הצפנה k, פעולות ההצפנה והפענוח מבוצעת באופן הבא: (קוד להמחשה בלבד)

```
for(i = 0; i < strlen(s); i++)

{

      s[i] = s[i] ^ k;

}
```

מימוש מנהל ההתקן

בתרגיל נממש מנהל התקן אשר יתמוך בשני סוגי התקני תווים, התקן מסוג Caesar Cipher והתקן מסוג XOR Cipher. עבור כל אחד משני סוגי ההתקנים, מנהל ההתקן ינהל חוצץ נתונים (Data Buffer) אשר אליו משתמש הקצה יכתוב מידע כדי להצפין מחרוזות, וממנו יקרא מידע כדי לפענח מחדש את המידע המוצפן.

בנוסף ליכולת לכתוב ולקרוא מההתקן, נרצה ששני ההתקנים יאפשרו למשתמש לבצע את הפעולות המיוחדות הבאות:

- Encryption Key:
  עלינו לאפשר למשתמש להגדיר את מפתח ההצפנה עימו יעשה שימוש כאשר המשתמש יכתוב אל ההתקן מידע כדי להצפינו, וכאשר המשתמש יקרא מן ההתקן מידע מוצפן כדי לפענחו. **שימו לב**, תכונה זו צריכה להיות מוגדרת ב- file objects אשר מתאר את הקשר העבודה של המשתמש עם ההתקן. למשל, אם פתחנו את אותו התקן (למשל, את Caesar Cipher) באמצעות שימוש בשתי קריאות open נפרדות (כך שכל אחת מהן החזירה לנו file descriptor שונה), אז יש לאפשר להגדיר encryption key נפרד עבור כל אחד מה- file objects שמיוצגים ע"י ה- file descriptors.
- Read State:
  עלינו לאפשר למשתמש להגדיר האם, כאשר הוא מבצע קריאה מן ההתקן, יקרא המידע המוצפן ללא פענוח (Raw Read) או שיקרא את המידע המוצפן לאחר פענוחו (Decrypt Read). **שימו לב**, גם תכונה זו צריכה להיות מוגדרת ב- file objects אשר מתאר את הקשר העבודה של המשתמש עם ההתקן.
- Zero Buffer:
  עלינו לאפשר למשתמש לבקש לאפס את חוצץ הנתונים של ההתקן.

אם כן, עליכם לכתוב Kernel Module אשר ירשום מנהל התקן חדש בעל מספר Major המוקצה דינאמית, ולממש שני File Operations Sets , אחד עבור Caesar Cipher אשר יוגדר ב- Minor 0, ואחד עבור XOR Cipher אשר יוגדר ב- Minor 1. להלן הנחיות כלליות למימוש ה- Kernel Module: **(ייתכן והמימוש שלכם יהיה שונה)**

הגדירו module parameter בשם memory_size

גודלם של שני החוצצים שיאכסנו את המחרוזת המוצפנת יקבע ע"י פרמטר חצוני של ה- kernel module בעת טעינתו.

ממשו את הפונקציה init_module

כפי שהוסבר בתרגול, פונקציה זו נקראת כאשר Kernel Module נטען לגרעין (למשל, ע"י הרצת הפקודה insmod מה- shell).

במימוש של פונקציה זו עליכם לבצע את הפעולות הבאות:

1. לרשום את מנהל ההתקן, ולשייך אותו למספר Major (ע"י קריאה ל- register_chrdev)
2. להקצות מקום לשני חוצצים אשר יאכסנו את המידע אשר נכתב אל שני ההתקנים (אחד עבור Caesar Cipher ואחד עבור XOR Cipher). גודלם של שני החוצצים יקבע ע"י הפרמטר memory_size.
3. שימו לב – כדי להקצות זכרון באופן דינאמי בתוך הקוד של הגרעין, יש להשתמש בפונקציה kmalloc. מידע נוסף על הפונקציה נמצא כאן.

ממשו את הפונקציה cleanup_module

כפי שהוסבר בתרגול, פונקציה זו נקראת כאשר Kernel Module נפרק מהגרעין (למשל, ע"י הרצת הפקודה rmmod מה- shell).

במימוש של פונקציה זו עליכם לבצע את הפעולות הבאות:

1. להסיר את מנהל ההתקן ממספר ה- Major אליו הוא משוייך (ע"י קריאה ל- unregister_chrdev)
2. למחוק את שני החוצצים שהוקצו עבור ההתקנים.
3. שימו לב – כדי לשחרר זיכרון שהוקצה באופן דינאמי בתוך הקוד של הגרעין, יש להשתמש בפונקציה kfree. מידע נוסף על הפונקציה נמצא כאן.

ממשו file operations עבור כל אחד משני ההתקנים

עליכם לממש את הפונקציות הבאות עבור כל התקן:

1. open
2. release
3. write
4. read
5. ioctl

אך שימו לב – למרות שלכל אחד משני ההתקנים יהיה אובייקט file_operations משלו, קל יותר לממש את התרגיל אם שניהם יחלקו את אותו המימוש עבור open, release ,ioctl משום שבשלושתן צריך לבצע את אותן הפעולות ללא קשר לסוג ההתקן.

ממשו את הפונקציה open

במימוש של פונקציה זו עליכם לבצע את הפעולות הבאות:

1. לבחור את אובייקט ה- file_operations המתאים (Caesar Cipher או XOR Cipher), בהתאם ל- minor ששמור ב- inode object.
2. להקצות מקום ב- private data של ה- file object, בו נאחסן את ה- key וה- read state (עליכם ליצור מבנה נתונים משלכם אשר יאחסן את שני השדות הנ"ל, ולגרום ל- private data להצביע על מופע שלו).

ממשו את הפונקציה release

במימוש של פונקציה זו עליכם לבצע את הפעולות הבאות:

1. לשחרר את המקום שהוקצה ב- private data של ה- file object.

ממשו את הפונקציה ioctl

במימוש של פונקציה זו עליכם לבצע את הפעולות הבאות:

1. לבדוק אם סוג הפקודה שנשלחה הינו change key. אם כן, אז יש לשמור בשדה key של ה- private data את ה- key החדש (גם סוג הפקודה וגם ערך מפתח ההצפנה החדש מתקבלים כפרמטרים של ioctl).
2. לבדוק אם סוג הפקודה שנשלחה הינו change read state. אם כן, אז יש לשמור בשדה read state של ה- private data את ה- read state החדש (גם סוג הפקודה וגם ערך סוג ה- read state החדש מתקבלים כפרמטרים של ioctl).
3. לבדוק אם סוג הפקודה שנשלחה הינו zero. אם כן, אז יש לאפס את חוצץ הנתונים של ההתקן. שימו לב, אם תהליך כלשהו פתח את אותו התקן מספר רב של פעמים (באמצעות קריאה ל- open מספר פעמים), אז כתוצאה מפעולה זו יתאפס החוצץ של ההתקן, ולכן, בניסיון לקרוא מן ההתקן, ע"י כל אחד מה- file descriptors שעובדים מולו, יוחזרו רק אפסים.

בקובץ encdec.h מוגדרים קבועים אשר עליכם לעשות בהם שימוש כדי להבדיל בין שתי הפקודות והארגומנטים שלהן. הקובץ מוגדר כך:

```
#ifndef _ENCDEC_H_

#define _ENCDEC_H_


#define ENCDEC_CMD_CHANGE_KEY      0

#define ENCDEC_CMD_SET_READ_STATE  1

#define ENCDEC_CMD_ZERO            2


#define ENCDEC_READ_STATE_RAW      0

#define ENCDEC_READ_STATE_DECRYPT  1


#endif
```

ממשו גרסה של write עבור Caesar Cipher
במימוש של פונקציה זו עליכם לבצע את הפעולות הבאות:

1. לכתוב את תוכן **החוצץ שהמשתמש העביר**, אל **החוצץ של התקן Caesar Cipher** באופן מוצפן (כמתואר לעיל). שימו לב, עליכם לכתוב את המידע החל מן המיקום הבא לכתיבה (המיקום הבא לכתיבה מתקבל כפרמטר loff_t *f_pos לפונקציה write).
2. אם לא ניתן לבצע כתיבה נוספת להתקן ההצפנה, משום ש- ערכו של f_pos* שווה ל- memory size של חוצץ ההצפנה של ההתקן, אז יש להחזיר את השגיאה ENOSPC-.
3. כאשר אתם מבצעים את העתקת המידע מהחוצץ של המשתמש (שנמצא ב- User space) אל החוצץ של ההתקן (שנמצא ב- Kernel space), עליכם להשתמש בפונקציה copy_from_user אשר מאפשרת להעתיק מידע מ- user space אל ה- kernel space בצורה בטוחה. מידע נוסף על פונקציה זו זמין בלינק הבא.
4. לאחר סיום הכתיבה, עליכם להוסיף ל- f_pos* את מספר התווים שכתבתם לחוצץ, כדי שבפעם הבאה שנכתוב מידע מוצפן אל החוצץ של **Caesar Cipher** נחל את הכתיבה מהמיקום הבא לכתיבה, ונמנע מדריסה של תווים שכבר נכתבו.
5. הפונקציה תחזיר את כמות הבתים שהצליחה לכתוב.
6. שימו לב שגם אם לא ניתן לכתוב את כל הבתים (התווים), נכתוב את המקסימום האפשרי ונחזיר את מספר הבתים שנכתב בפועל.
7. **שימו לב – גם אם אותו התקן נפתח ע"י תהליך מספר רב של פעמים (ע"י קריאה ל- open מספר פעמים), כל פעולות הכתיבה (write) שיופנו ל- file descriptors שהתקבלו כתוצאה מקריאה ל- open, המייצגים בסופו של דבר את אותו ההתקן, יכתבו כולן אל אותו החוצץ (משום שמנהל ההתקן (ה- driver) מתחזק רק שני חוצצים, אחד לכל התקן).**

ממשו גרסה של write עבור XOR Cipher
עקבו אחר השלבים שתוארו לעיל עבור write של Caesar Cipher, רק שבמקום לבצע את ההצפנה באמצעות Caesar Cipher בצעו אותה באמצעות XOR.

ממשו גרסה של read עבור Caesar Cipher
במימוש של פונקציה זו עליכם לבצע את הפעולות הבאות:

1. לקרוא את תוכן **החוצץ של התקן Caesar Cipher**, אל **החוצץ של המשתמש** באופן **מפוענח** (decrypted) או **חשוף** (raw), בהתאם ל- read state שנמצא ב- private data של ה- file object. שימו לב, עליכם לקרוא את המידע החל מן המיקום הבא לקריאה (המיקום הבא לקריאה מתקבל כפרמטר loff_t *f_pos לפונקציה read).
2. אם לא ניתן לבצע קריאה נוספת מהתקן ההצפנה, משום ש- ערכו של f_pos* שווה ל- memory size של חוצץ ההצפנה של ההתקן, אז יש להחזיר את השגיאה EINVAL-.
3. כאשר אתם מבצעים את העתקת המידע מהחוצץ של ההתקן (שנמצא ב- Kernel space) אל החוצץ של המשתמש (שנמצא ב- User space), עליכם להשתמש בפונקציה copy_to_user אשר מאפשרת להעתיק מידע מ- kernel space אל ה- user space בצורה בטוחה. מידע נוסף על פונקציה זו זמין בלינק הבא.
4. לאחר סיום הקריאה, עליכם להוסיף ל- f_pos* את מספר התווים שנקראו לתוך החוצץ, כדי שבפעם הבאה שנקרא מידע מהחוצץ של ההתקן אל החוצץ של המשתמש, נחל את הקריאה מהמיקום הבא לקריאה, ונמנע מקריאת אותו המידע פעם נוספת.
5. הפונקציה תחזיר את כמות התווים שהיא הצליחה לקרוא
6. שימו לב שגם אם החוצץ מכיל כמות יותר קטנה מהכמות הנדרשת, על הפונקציה לאפשר את זה ולהחזיר את הכמות של בתים (תווים) שהיא הצליחה לקרוא
7. **שימו לב – גם אם אותו התקן נפתח ע"י תהליך מספר רב של פעמים (ע"י קריאה ל- open מספר פעמים), כל פעולות הקריאה (read) שיופנו ל- file descriptors שהתקבלו כתוצאה מקריאה ל- open, המייצגים בסופו של דבר את אותו ההתקן, יקראו כולן מאותו החוצץ (משום שמנהל ההתקן (ה- driver) מתחזק רק שני חוצצים, אחד לכל התקן).**

ממשו גרסה של read עבור XOR Cipher
עקבו אחר השלבים שתוארו לעיל עבור read של Caesar Cipher, רק שבמקום לבצע את הפענוח באמצעות Caesar Cipher בצעו אותה באמצעות XOR.

קומפילציה, הרצה ובדיקה

בקובץ ה- ZIP של התרגיל מסופקים לכם הקבצים הבאים:

1. encdec.c
2. encdec.h (אסור לערוך קובץ זה)
3. Makefile (אסור לערוך קובץ זה)
4. load (אסור לערוך קובץ זה)
5. unload (אסור לערוך קובץ זה)
6. test (אסור לערוך קובץ זה)
7. test.c (אסור לערוך קובץ זה)
8. test1.in, test2.in, test3.in, test4.in, test5.in
9. test1.out, test2.out, test3.out, test4.out, test5.out

אנא וודאו כי כל הקבצים נמצאים באותה הספרייה כאשר אתם עובדים על התרגיל.

להלן הסבר על כל אחד מן הקבצים:

1. הקובץ encdec.c הינו קוד שלד שמהווה נקודת התחלה לכתיבת התרגיל – השתמשו בו.
2. הקובץ encdec.h הינו קובץ המכיל הגדרות של קבועים אשר ישמשו אתכם במימוש התרגיל – אסור לבצע שינויים בקובץ זה, וגם אין להגישו – קובץ זה יתווסף אוטומטית כאשר נבדוק את הגשותיכם.
3. הקובץ Makefile הינו קובץ המגדיר כיצד להדר את התרגיל ע"י GCC. כדי להדר את המודול שכתבתם, עליכם להריץ את הפקודה make באמצעות ה- shell. כתוצאה מפקודה זו, GCC יהדר את המודול שכתבתם באמצעות ההגדרות שנמצאות ב- Makefile.
4. הקובץ load הינו shell script אשר הרצה שלו מבצעת שני דברים:
   a. טוענת את המודול שהידרתם (ע"י פקודת make) לתוך הגרעין של לינוקס (ע"י הרצה של הפקודה insmod).
   b. יוצרת שני קבצי התקנים – קובץ התקן אחד תחת בכתובת "/dev/encdec0" בעל minor=0, וקובץ התקן שני תחת הכתובת "/dev/encdec1" בעל minor=1.

   שימו לב – load מקבלת פרמטר בשם memory_size אשר בהמשך מועבר כפרמטר למודול שכתבתם. כלומר, אם תריצו את הפקודה "./load memory_size=100" אז load תאתחל את המודול שכתבתם, כך שהפרמטר memory_size של יהיה 100.

5. הקובץ unload הינו shell script אשר הרצה שלו מבצעת את הפעולה ההפוכה של load:
   a. מסירה את המודול שנטען ע"י load מהגרעין של לינוקס (ע"י הרצה של הפקודה rmmod).
   b. מוחקת את שני קבצי ההתקנים שנוצרו ע"י load.
6. הקובץ test הינה תוכנית אשר באמצעותה תבדקו את תקינות מנהל ההתקן שתכתבו. התוכנית מאפשרת למשתמש להזין את הפקודות הבאות:
   a. **`open #device_id #reference_id #flags`**

   פקודה זו תגרום לתהליך לקרוא לפונקציה open עבור פתיחת התקן. להלן פירוט:
      i. #device_id יכול להיות הערך 0 או הערך 1. עבור ערך 0 ייפתח ההתקן תחת הכתובת "/dev/encdec0" ועבור הערך 1 ייפתח ההתקן תחת הכתובת "/dev/encdec1".
      ii. #reference_id הינו מספר המשויך ל- file descriptor האמיתי שהתקבל כתוצאה לקריאה לפונקציה open.
      iii. #flags יכול להיות אחד מהערכים הבאים – "read", "write" או "read|write".

   פקודה לדוגמה –`open 0 2 read`

   משמעות פקודה זו תהיה לפתוח את התקן "/dev/encdec0" לקריאה, ולשייך את ה- file descriptor שיוחזר ע"י הקריאה ל- open, למספר 2.

   b. **`write #reference_id "#string"`**

   פקודה זו תגרום לתהליך לקרוא לפונקציה write עבור כתיבה להתקן. להלן פירוט:
      i. #reference_id הינו מספר הייחוס עבור ה- file descriptor אליו נבצע את הכתיבה.
      ii. #string המחרוזת אותה אנו מעוניינים לכתוב להתקן (כדי להצפינה).

פקודה לדוגמה – `write 1 "Hello World"`
משמעות פקודה זו תהיה לכתוב את המחרוזת "Hello World" אל ה- file descriptor שמזוהה עם המספר 1.

c. **read #reference_id #count**
פקודה זו תגרום לתהליך לקרוא לפונקציה read לשם קריאה מהתקן. להלן פירוט:
   i. #reference_id הינו מספר הייחוס עבור ה- file descriptor ממנו נקרא נתונים.
   ii. #count מספר התווים אותו אנו מעוניינים לקרוא.

פקודה לדוגמה – `read 1 5`
משמעות פקודה זו תהיה לקרוא 5 תווים מה- file descriptor שמזוהה עם המספר 1.

d. **lseek #reference_id #pos**
פקודה זו תגרום לתהליך לקרוא לפונקציה lseek בכדי להזיז את ה- seek pointer של ה- file descriptor שמזוהה עם ה- reference id הנתון. להלן פירוט:
   i. #reference_id הינו מספר הייחוס עבור ה- file descriptor עליו נפעיל את lseek
   ii. #pos המיקום החדש של ה- seek pointer.

פקודה לדוגמה – `lseek 0 0`
משמעות פקודה זו תהיה להזיז את ה- seek pointer של ה- file descriptor שמזוהה עם המספר 0, כך שיצביע על התו הראשון בחוצץ של ההתקן.

e. **ioctl #reference_id #cmd #arg**
פקודה זו תגרום לתהליך לקרוא לפונקציה ioctl עבור ה- file descriptor המזוהה עם ה- reference id הנתון.
   i. #reference_id הינו מספר הייחוס עבור ה- file descriptor עליו נפעיל את ioctl
   ii. #cmd מזהה הפקודה שברצוננו לבצע – יכול להיות אחת משלושת האפשרויות הבאות "change_key", "change_read_state" או "zero".
   iii. #arg ארגומנט הפקודה שברצוננו לספק. נדרש במקרים הבאים:
      1. אם #cmd = change_key. במקרה זה, יכיל את ערכו של מפתח ההצפנה החדש.
      2. אם #cmd = change_read_state. במקרה זה, יכיל את הערך decrypt או raw.

פקודה לדוגמה – `ioctl 2 change_key 5`
פקודה לדוגמה – `ioctl 2 change_read_state raw`

f. **close #reference_id**
פקודה זו תגרום לתהליך לקרוא לפונקציה close עבור ה- file descriptor המזוהה עם ה- reference id הנתון.
   i. #reference_id הינו מספר הייחוס עבור ה- file descriptor עליו נפעיל את close

g. **exit**
מסיים את ריצת התוכנית.

7. הקובץ test.c הינו קוד המקור של התוכנית test – אנא בדקו כי אתם מבינים כיצד הפונקציה execute_command בקוד עובדת (זה יעזור להבנת התרגיל).

אופן עבודה

כאשר תרצו לבדוק את המודול שכתבתם, עליכם לעבוד באופן הבא:

1. הריצו את התוכנית unload כדי להסיר את גרסת המודול הנוכחית שמותקנת בגרעין של לינוקס.
2. הריצו את התוכנית make כדי להדר מחדש את המודול.
3. הריצו את התוכנית load כדי לטעון מחדש את המודול המהודר.
4. הריצו את התוכנית test כדי לעבוד מול מנהל ההתקן שהותקן ע"י המודול. באפשרותכם להקליד לתוך התוכנית test את הפקודות באופן ידני, או לכתוב קבצי בדיקה משלכם, ולהפנות אותם כקלט לתוכנית test ע"י input redirection.
5. **לנוחיותכם, ניתן לראות <u>בוידאו הבא</u> דוגמה לשימוש באופן העבודה המפורט לעיל.**

הגשה

ההגשה הינה אלקטרונית דרך Moodle. עקבו אחר השלבים הבאים:

1. עליכם ליצור קובץ zip (השתמשו ב-zip או gzip בלבד) בשם hw2_id1_id2 כאשר id1,id2 מייצגים את מספרי תעודות הזהות של המגישים.
2. תכולת קובץ ה zip צריכה להיות התכולה הבאה (ללא תתי ספריות!):
   - encdec.c
   - קובץ בשם submitters.txt שמכיל את מספרי הזהות והשמות של מגישי התרגיל מופרדים על ידי פסיק במבנה הבא (לדוגמה):

```
Bill Gates,bill@microsoft.com,123456789

Linus Torvalds,linus@gmail.com,234567890
```

3. את קובץ ה- zip יש ליצור ע"י הרצת הפקודה הבאה:

```
zip hw3_id1_id2.zip encdec.c submitters.txt
```

4. הגישו את קובץ ה- zip דרך Moodle.