RSA Encryption and Signature Lab

מבוא:

RSA (Rivest–Shamir–Adleman) היא אחת ממערכות ההצפנה הציבוריות הראשונות והיא נמצאת בשימוש נרחב לאבטחה
תקשורת. אלגוריתם RSA יוצר תחילה שני מספרים ראשוניים אקראיים גדולים, ולאחר מכן משתמש בהם
כדי ליצור זוגות מפתחות ציבוריים ופרטיים, אשר ניתן להשתמש בהם כדי לבצע הצפנה, פענוח, חתימה דיגיטלית
יצירת ואימות חתימה דיגיטלית. אלגוריתם RSA בנוי על תאוריות מספרים, והוא יכול להיות מיושם די בקלות עם תמיכה של ספריות.
מטרת הלמידה של מעבדה זו היא לצבור התנסויות מעשיות באלגוריתם RSA.
אלגוריתם RSA כולל חישובים על מספרים גדולים. לא ניתן לרמות חישובים אלה באופן ישיר
באמצעות אופרטורים אריתמטיים פשוטים בתוכניות, מכיוון שמפעילים אלה יכולים לפעול רק על פרימיטיביים
סוגי נתונים, כגון סוגים שלמים של 32 סיביות ומספרים שלמים באורך 64 סיביות.
המספרים המעורבים באלגוריתמים של RSA
הם בדרך כלל יותר מ 512 סיביות. לדוגמה, כדי להכפיל שני מספרים שלמים של 32 סיביות a ו- b, אנו פשוט
צריך להשתמש ב- A * B בתוכנית שלנו. עם זאת, אם מדובר במספרים גדולים, איננו יכולים לעשות זאת יותר; במקום
אנחנו צריכים להשתמש באלגוריתם (כלומר, פונקציה) כדי לחשב את המוצרים שלהם.
ישנן מספר ספריות שיכולות לבצע פעולות אריתמטיות על מספרים שלמים בגודל שרירותי. בזה
מעבדה, נשתמש בספריית המספרים הגדולים המסופקת על ידי openSSL. כדי להשתמש בספרייה זו, נגדיר כל גדול
מספר כסוג BIGNUM ולאחר מכן השתמש בממשקי ה- API שסופקו על-ידי הספריה עבור פעולות שונות, כגון
חיבור, כפל, אקספוננציה, פעולות מודולריות וכו '.

Task 1: Deriving the Private Key

RSA היא מערכת הצפנה מבוססת על מפתחות ציבוריים ופרטיים.
במטלה זו נקבל 3 מספרים ראשוניים e, q, p ונשתמש באלגוריתם על מנת לפתח מהם את המפתח הפרטי. כדי לבצע זאת נצטרך להשתמש בספרייה openssl/bn על מנת שנוכל לבצע את החישובים מכיוון שאורך המספרים הוא שנשתמש בהם הוא 128 סיביות.
חישוב n = p*q
חישוב (1-q-1)*(p)

בנינו סקריפט ובו בנינו את המפתח הפרטי בהינתן המספקים p,q,e

```
#include <stdio.h>
#include <openssl/bn.h>
void printBN(char *msg, BIGNUM * a)
{
/* Use BN_bn2hex(a) for hex string
* Use BN_bn2dec(a) for decimal string */
char * number_str = BN_bn2hex(a);
printf("%s %s\n", msg, number_str);
OPENSSL_free(number_str);
}
int main(){

        BN_CTX *ctx = BN_CTX_new();

        BIGNUM *p = BN_new();
        BIGNUM *q = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *e = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *one = BN_new();
        BIGNUM *p_minus_one = BN_new();
        BIGNUM *q_minus_one = BN_new();
        BIGNUM *tot_n = BN_new();

        BN_dec2bn(&one, "1");
        BN_hex2bn(&e, "0D88C3");
        BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
        BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");

        // calculate n
        BN_mul(n, p, q, ctx);
```

תחילה ייבאנו את הספריות החשובות ולאחר מכן הצהרנו על כל המשתנים שאנחנו זקוקים להם על מנת לחשב את המפתח הפרטי d.
בשורה האחרונה של הקוד חישבנו את n = p*q.

לאחר שמצאנו את n חישבנו tot_n ויצרנו מהנתונים את המפתח הפרטי d

```
// calculate n-(p-q) * (q-1)
BN_sub(p_minus_one, p, one);
BN_sub(q_minus_one, q, one);
BN_mul(tot_n, p_minus_one, q_minus_one, ctx);

// calculate private key
BN_mod_inverse(d, e, tot_n, ctx);

printBN("private key = ", d);
```

את tot_n חישבנו בעזרת : (tot_n = (p-1)*(q-1
ולאחר מכאן השתמשנו בפונקציה של openssl/bn על מנת לייצר את המפתח הפרטי d.
הרצנו את הקוד:

```
[06/06/2024 12:39] Attacker >>> sudo vi deriving_private_key.c
[06/06/2024 12:40] Attacker >>> gcc deriving_private_key.c -lcrypto
 -o diriving_private_key
[06/06/2024 12:41] Attacker >>> ./diriving_private_key
private key =  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7
C28A9B496AEB
```

וקיבלנו את המפתח הפרטי הנ"ל.
סיכום:

1. מאתחלים את המשתנים הדרושים לחישוב: p, q, e, n, d, totient והBN_CTX.
2. מחשבים את n = p * q ו־(1-totient = (p-1) * (q.
3. משתמשים בBN_mod_inverse־ כדי לחשב את המפתח הפרטי d על פי המספרים הראשוניים p, q ו־e.
4. מדפיסים את המפתח הפרטי d בפורמט hex ומשחררים משאבים כדי למנוע דליפת זיכרון.

בכך, אנו מסיימים את פיתוח המפתח הפרטי למערכת RSA על פי הקלט שנקבל.

Task 2: Encrypting a Message

במטלה זו עלינו להשתמש במפתח ציבורי (e, n) על מנת להצפין את ההודעה " A top secret".

ובעזרת המפתח הפרטי d לפענח את ההצפנה.

כתבנו קוד ב C שמצפין את ההודעה.

```
#include <stdio.h>
#include <openssl/bn.h>
void printBN(char *msg, BIGNUM * a)
{
/* Use BN_bn2hex(a) for hex string
* Use BN_bn2dec(a) for decimal string */
char * number_str = BN_bn2hex(a);
printf("%s %s\n", msg, number_str);
OPENSSL_free(number_str);
}

int main(){

        BN_CTX *ctx = BN_CTX_new();

        BIGNUM *e = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *M = BN_new();
        BIGNUM *C = BN_new();

        BIGNUM *M_Decrypt = BN_new();

        // given value for encode in hex "A top secret!"
        BN_hex2bn(&M, "4120746f702073656372657421");
        BN_hex2bn(&e, "010001");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E479414
8AACBC26AA381CD7D30D");
        BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3
A4D0CB81629242FB1A5");

        // calculate C = M^c mod n
        BN_mod_exp(C,M,e,n,ctx);
        // calculate M_Decrypt = C^d mod n
        BN_mod_exp(M_Decrypt, C, d , n ,ctx);
"encrypting_a_message.c" 42L, 1071C              1,1           Top
```

תחילה כמובן הצהרנו על כל משתנים שאנחנו צריכים בשביל לבצע את ההצפנה.

לאחר מכן על מנת להצפין את M נשתמש בחישוב C=(M^c)%n

וכדי לבטל את ההצפנה C כדי לקבל שוב את M נשתמש בחישוב : M=(C^d)%n

בקטע הקוד הזה נבצע בדיקה אם ההצפנה והפענוח הצליחו.

```
printBN("cryphertext (c) = ", C);

if(BN_cmp(M_Decrypt, M) == 0)
        printf("encryption succeed \n");
else
        printf("encryption failed \n");
```

על מנת לוודא שהצלחנו השוונו את ההודעה עם הפיצוח של ההודעה המוצפנת

הרצנו את הקוד:

```
[06/06/2024 12:47] Attacker >>> sudo vi encrypting_a_message.c
[06/06/2024 12:57] Attacker >>> gcc encrypting_a_message.c -lcrypto
 -o encrypting_a_message
[06/06/2024 12:57] Attacker >>> ./encrypting_a_message
cryphertext (c) =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75C
ACDC5DE5CFC5FADC
encryption succeed
```

אפשר לראות שהצלחנו מכיוון שהודפס encryption succeed כלומר ההודעה ופענוח הצופן היו זהים.

סיכום:

במטלה זו, פיתחנו קוד בשפת C להצפנה ופענוח באמצעות מערכת RSA, שבה משתמשים במפתח ציבורי (e, n) להצפנת הודעה ובמפתח פרטי d לפענוחה. הקוד בנה על ספריית OpenSSL לטיפול במספרים גדולים ובפעולות מתמטיות מורכבות כמו חזקות וחילוק במרחב מודולרי.

השלבים העיקריים בקוד כללו:

1. יצירת מפתחות ציבורי ופרטי: הגדרת ערכי p, q ו-e, חישוב n ו־totient, וחישוב d כמפתח פרטי באמצעות BN_mod_inverse.
2. הצפנה: השתמשנו בחישוב C = (M^e) % n על מנת להצפין את הודעת " A top secret".
3. פענוח: השתמשנו בחישוב M = (C^d) % n על מנת לקבל את הודעת המקור.

בסופו של דבר, ביצענו בדיקת תקינות על ידי השוואת ההודעה המקורית עם ההודעה המפוענחת, ואימתנו שהצפנה והפענוח היו מוצלחים. זה הדגיש את יכולת המערכת להצפין ולפענח מידע באמצעות שימוש במפתחות ציבוריים ופרטיים שנוצרו במהלך התהליך.

Task 3: Decrypting a Message

במטלה זו קיבלנו צופן C וננסה באמצעות המפתח הפרטי להמיר אותו בחזרה אל הטקסט המקורי

כתבנו קוד ב C שמפענח את הצופן:

```
#include <openssl/bn.h>
void printBN(char *msg, BIGNUM * a)
{
/* Use BN_bn2hex(a) for hex string
* Use BN_bn2dec(a) for decimal string */
char * number_str = BN_bn2hex(a);
printf("%s %s\n", msg, number_str);
OPENSSL_free(number_str);
}

int main(){

        BN_CTX *ctx = BN_CTX_new();

        BIGNUM *e = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *M = BN_new();
        BIGNUM *C = BN_new();

        BIGNUM *M_Decrypt = BN_new();

        BN_hex2bn(&e, "010001");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E479414
8AACBC26AA381CD7D30D");
        BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3
A4D0CB81629242FB1A5");
        BN_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7
DCB67396567EA1E2493F");

        // calculate the message
        BN_mod_exp(M, C, d, n, ctx);

        printBN("Message (M) = ",M);
};
```

תחילה הצהרנו על המשתנים שהקוד צריך להשתמש בהם על מנת לפצח את הצופן

כדי לקבל את ההודעה M מה צופן C נשתמש שוב בחישוב: M=(C^d)%n

הרצנו את הקוד שכתבנו:

```
[06/06/2024 13:00] Attacker >>> sudo vi decrypting_a_message.c
[06/06/2024 13:03] Attacker >>> gcc decrypting_a_message.c -lcrypto
 -o decrypting_a_message
[06/06/2024 13:04] Attacker >>> ./decrypting_a_message
Message (M) =  50617373776F72642069732064656573
```

וקיבלנו את ההודעה בבסיס 16

המרנו את הטקסט שקיבלנו בבסיס 16 אל ASCII:

```
[06/06/2024 13:07] Attacker >>> python -c 'print("50617373776F72642
069732064656573".decode("hex"))'
Password is dees
```

הצלחנו מכיוון שקיבלנו הודעה תקינה ולו טעות אחת לא היינו מקבלים את ההודעה הזו.

סיכום:

הצפן C בחזרה להודעה המקורית M.

השלבים העיקריים בקוד כללו:

1. יצירת משתנים ואתחולם: הגדרת ערכי d, n ו־C, המייצג את הצפן שנרצה לפענח.
2. פענוח הצפן: השתמשנו בחישוב M = (C^d) % n על מנת להמיר את הצפן C בחזרה להודעה המקורית M.
3. המרת M מבסיס 16 לטקסט ASCII: בסיום החישובים, המרנו את התוצאה שקיבלנו מבסיס 16 לטקסט בקוד ASCII.

בביצוע הקוד, אימתנו שהפענוח הצליח על ידי השוואת ההודעה המקורית שקיבלנו עם ההודעה שיצאה מהפענוח. הצלחנו להשיג תוצאה תקינה, מה שמעיד על תקינות המפתח הפרטי שהשתמשנו בו ועל היכולת של הקוד לפענח את הצפן באופן נכון.

בסיכום, הקוד שפיתחנו מוכיח את תקינות המפתח הפרטי שנוצר ואת יכולתו להפוך את הצפן C להודעה מובנת בצורה מדויקה ובלתי תלויתית מהאופן שבו התקבלה ההצפנה

Task 4: Signing a Message

במטלה זו התבקשנו לבצע שינוי בהודעה " I owe you $2000". ולחפש את ההבדלים בין ערכי ההודעה בבסיס 16
ולאחר מכן לחפש את ההבדלים בין החתימות על ההודעות עצמן.

המרנו בעזרת קוד C את ההודעות אל בסיס 16

```
[06/06/2024 13:14] Attacker >>> python -c 'print("I owe you $2000."
.encode("hex"))'
49206f776520796f752024323030302e
[06/06/2024 13:16] Attacker >>> python -c 'print("I owe you $3000."
.encode("hex"))'
49206f776520796f752024333030302e
```

אפשר לראות את ההבדל באינדקס ה 8 מהסוף ב $2000 יש את הספרה 2 וב $3000 יש את הספרה 3 כפי שממורקר.

כתבנו קוד ב C שיצפין את ההודעות:

```
#include <stdio.h>
#include <openssl/bn.h>
void printBN(char *msg, BIGNUM * a)
{
/* Use BN_bn2hex(a) for hex string
* Use BN_bn2dec(a) for decimal string */
char * number_str = BN_bn2hex(a);
printf("%s %s\n", msg, number_str);
OPENSSL_free(number_str);
}
```

המשך הקוד ב C:

```c
int main(){

        BN_CTX *ctx = BN_CTX_new();

        BIGNUM *e = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *M2000 = BN_new();
        BIGNUM *M3000 = BN_new();
        BIGNUM *S3000 = BN_new();
        BIGNUM *S2000 = BN_new();


        BN_hex2bn(&e, "010001");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
        BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

        // value from printing the message "I owe you $2000
        BN_hex2bn(&M2000, "49206f776520796f752024323030302e");
        // value from printing the message "I owe you $3000
        BN_hex2bn(&M3000, "49206f776520796f752024333030302e");

        BN_mod_exp(S2000, M2000, d, n, ctx);
        BN_mod_exp(S3000, M3000, d, n, ctx);

        printBN("Signature (S2000) = ", S2000);
        printBN("\nSignature (S3000) = ", S3000);

};
```

תחילה הצהרנו על המשתנים שהקוד צריך בשביל לעבוד והשתמשנו בחישוב C=(M^c)%n על מנת להצפין את שתי ההודעות.

הרצנו את הקוד שכתבנו

```
[06/06/2024 13:30] Attacker >>> sudo vi signing_a_message.c
[06/06/2024 13:30] Attacker >>> gcc signing_a_message.c -lcrypto -o signing_a_message
[06/06/2024 13:31] Attacker >>> ./signing_a_message
Signature (S2000) =  55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB

Signature (S3000) =  BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
[06/06/2024 13:31] Attacker >>>
```

אפשר לראות שתוצאות ההצפנה הן שונות מאוד גם כאשר נעשה שינוי מינורי בהודעה עצמה ולא יהיה ניתן לשחזר את ההודעה כמו בהמרת ההודעה לבסיס 16.

סיכום:

במטלה זו, ביצענו שינוי בהודעה מקורית "I owe you $2000" להודעה חדשה " I owe you $3000", ואחר כך ביצענו הצפנה לשתי ההודעות באמצעות מערכת RSA עם מפתח ציבורי קבוע (e, n).

השלבים העיקריים בקוד כללו:

1. הצהרת משתנים ואתחולם: הגדרת ערכי e, n, והודעות המקור והמשונות.
2. הצפנה: שימוש בחישוב C = (M^e) % n כדי להצפין את שתי ההודעות.
3. השוואת ההבדלים: השוואה בין ערכי ההודעות המקוריות בבסיס 16, ובין חתימותיהן של ההודעות עצמן.

בביצוע הקוד גילינו שהתוצאות של ההצפנה שונות באופן מוחלט בין ההודעה המקורית לבין ההודעה המשונתה. זה מדגיש את עובדה שהצפנה במערכת RSA היא תקינה ושינוי קטן בהודעה מביא לתוצאה שונה לחלוטין בצפן.

המטלה הדגישה את חוזקם של מערכות הצפנה מבוססות על מפתחות ציבוריים ופרטיים כמו RSA, שמבטיחות עמידות למידע, תוך הדגשה על החודש החשיבותי של יציבות המפתחות בתהליך ההצפנה והפענוח.

Task 5: Verifying a Signature

מבוא:

במקרה הזה, עלינו לאמת חתימה דיגיטלית שנשלחה מאת אליס אל בוב. החתימה הדיגיטלית נועדה לוודא שההודעה שנשלחה אכן הגיעה מאליס ולא שונתה בדרך.

להלן המרכיבים שיש לנו:

M: ההודעה המקורית שהתקבלה על ידי בוב, לפני שהיא קודדה:

S: החתימה הדיגיטלית שהתקבלה מאת אליס, בקידוד הקסדצימלי:

e: האקספוננט הציבורי במפתח הציבורי של אליס (בקידוד הקסדצימלי):

n: המודולוס במפתח הציבורי של אליס (בקידוד הקסדצימלי):

המפתח הציבורי של אליס מורכב מהזוג (e, n).

כדי לאמת את החתימה, יש לבצע את השלבים הבאים:

1. המרת החתימה S למספר שלם (Integer).
2. ביצוע חישוב של Semod nS^e \mod nSemodn בעזרת המפתח הציבורי של אליס (e, n).
3. המרה חזרה של התוצאה מהשלב הקודם להודעה המקורית ובדיקה אם היא תואמת להודעה M שקיבלנו.

נראה את הערך ב hex של ההודעה launch a missle.

```
[06/06/2024 13:36] Attacker >>> python -c 'print("Launch a missile.
".encode("hex"))'
4c61756e63682061206d697373696c652e
```

הקוד בפייתון שמכיל את כל הנתונים שציינו למעלה וחישוב ה Hash

```c
int main(){

        BN_CTX *ctx = BN_CTX_new();

        BIGNUM *e = BN_new();
        BIGNUM *M = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *H = BN_new();
        BIGNUM *S = BN_new();

        BN_hex2bn(&e, "010001");

        BN_hex2bn(&M,"4c61756e63682061206d697373696c652e");
        BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595
113AA51B450F18116115");
        BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005
CAB026C0542CBDB6802F");

        // calculate H = S^e mod n
        BN_mod_exp(H, S, e, n, ctx);

        printBN("Hash (H) = ", H);
        printBN("Message (M) = ", M);

        if (BN_cmp(H,M) == 0)
                printf("Alice sent the message\n");
        else
                printf("Alice NOT sent the message \n");
};
```

תוצאות הקוד.

```
[06/06/2024 13:43] Attacker >>> sudo vi verifying_a_signature.c
[06/06/2024 13:44] Attacker >>> gcc verifying_a_signature.c -lcrypt
o -o verifing_a_signature
[06/06/2024 13:44] Attacker >>> ./verifing_a_signature
Hash (H) =  4C61756E63682061206D697373696C652E
Message (M) =  4C61756E63682061206D697373696C652E
Alice sent the message
```

לאחר ביצוע הקוד כפי שניתן לראות ה decode של ההודעה וקבלת ה HASH זהה ל Hash של ההודעה.

נבצע בדיוק את אותו הדבר, רק שהפעם נשנה את הסיומת של החתימה הדיגיטלית מ F2 ל F3

```
int main(){

        BN_CTX *ctx = BN_CTX_new();

        BIGNUM *e = BN_new();
        BIGNUM *M = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *H = BN_new();
        BIGNUM *S = BN_new();

        BN_hex2bn(&e, "010001");

        BN_hex2bn(&M,"4c61756e63682061206d697373696c652e");
        BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595
113AA51B450F18116115");
        BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005
CAB026C0542CBDB6803F");

        // calculate H = S^e mod n
        BN_mod_exp(H, S, e, n, ctx);

        printBN("Hash (H) = ", H);
        printBN("Message (M) = ", M);

        if (BN_cmp(H,M) == 0)
                printf("Alice sent the message\n");
        else
                printf("Alice NOT sent the message \n");
};
```

נריץ ונראה את התוצאה

```
[06/06/2024 13:45] Attacker >>> sudo vi verifying_a_signature.c
[06/06/2024 13:47] Attacker >>> gcc verifying_a_signature.c -lcrypt
o -o verifing_a_signature
[06/06/2024 13:47] Attacker >>> ./verifing_a_signature
Hash (H) =  91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE020
3B41AC294
Message (M) =  4C61756E63682061206D697373696C652E
Alice NOT sent the message
```

כמו שציפינו, לאחר ששינינו את הסיומת של S מ F2 ל F3 אכן ה Hash לא היה זהה.

סיכום:

במטלה זו, עבדנו עם חתימה דיגיטלית שנשלחה מאליס אל בוב, ועברנו תהליך אימות חתימה דיגיטלית כדי לוודא שההודעה שנשלחה לא השתנתה בדרך והגיעה באמת מאליס.

השלבים העיקריים שביצענו כללו:

1. המרת החתימה S שקיבלנו מאליס מהקידוד ההקסדצימלי למספר שלם (Integer).
2. ביצוע החישוב S^e % n על פי המפתח הציבורי של אליס (e, n) כדי לקבל Semod n.
3. המרה של התוצאה מספרית חזרה לטקסט וביצוע HASH ביצוע ומיוצר.

הבדיקה התבצעתה ונראה כי כאשר שינינו את החתימה הדיגיטלית על מנת להוסיף את הסיומת S מ F2 ל F3, התוצאה של HASH של ההודעה לא היתה תואמת להודעה המקורית. זה מצביע על חוזק המערכת בלתי תלויתיות לבדיקות של נתונים במערכת.

Task 6: Manually Verifying an X.509 Certificate

בתרגיל זה נבחר את www.facebook.com. לאחר הרצת הפקודה של openssl s_client -connect www.facebook.com:443 -showcerts נקבל C1, C2 ונשמור אותם כקבצי pem

פלט לאחר כתיבת הפקודה:

```
[06/07/2024 11:54] Attacker >>> openssl s_client -connect www.faceb
ook.com:443 -showcerts
CONNECTED(00000003)
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiC
ert High Assurance EV Root CA
verify return:1
depth=1 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiC
ert SHA2 High Assurance Server CA
verify return:1
depth=0 C = US, ST = California, L = Menlo Park, O = "Meta Platform
s, Inc.", CN = *.facebook.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=California/L=Menlo Park/O=Meta Platforms, Inc./CN=*.f
acebook.com
   i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 High
 Assurance Server CA
-----BEGIN CERTIFICATE-----
MIIGljCCBX6gAwIBAgIQDwR46ni8deWiq0M8WBSHCzANBgkqhkiG9w0BAQsFADBw
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMS8wLQYDVQQDEyZEaWdpQ2VydCBTSEEyIEhpZ2ggQXNz
dXJhbmNlIFNlcnZlciBDQTAeFw0yNDAzMTYwMDAwMDBaFw0yNDA2MTQyMzU5NTla
MG8xCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRMwEQYDVQQHEwpN
ZW5sbyBQYXJrMR0wGwYDVQQKExRNZXRhIFBsYXRmb3JtcywgSW5jLjEXMBUGA1UE
AwwOKi5mYWNlYm9vay5jb20wWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATMfb1I
B3qERPRrIQX8oewiMnP6cIC1DaGAfMEuSLZnJU9sLUnAtlnmvG+AQE7V51IF07WK
1W1cSbjQsmSHLqr6o4ID9jCCA/IwHwYDVR0jBBgwFoAUUWj/kK8CB3U8zNllZGKi
ErhZcjswHQYDVR0OBBYEFMrtwob6p4ZcL4Sjf42bOgIMGzXKMIG1BgNVHREEga0w
```

נשמור את ה certeficate הראשון בקובץ c1.pem

c1:

```
[06/06/2024 14:03] Attacker >>> cat c1.pem
-----BEGIN CERTIFICATE-----
MIIGlzCCBX+gAwIBAgIQCxBY53h8TmDGxc0Lg8LaDzANBgkqhkiG9w0BAQsFADBw
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMS8wLQYDVQQDEyZEaWdpQ2VydCBTSEEyIEhpZ2ggQXNz
dXJhbmNlIFNlcnZlciBDQTAeFw0yNDAzMTUwMDAwMDBaFw0yNDA2MTMyMzU5NTla
MG8xCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRMwEQYDVQQHEwpN
ZW5sbyBQYXJrMR0wGwYDVQQKExRNZXRhIFBsYXRmb3JtcywgSW5jLjEXMBUGA1UE
AwwOKi5mYWNlYm9vay5jb20wWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAT9Dt5Q
DzXN8whYBFI3kPa7eS0dYJKLhKf8aBy5TiQDA3fJNcLhfNdT+fJ3KsapsDFMbj1T
zPCG93igTFTFyrHpo4ID9zCCA/MwHwYDVR0jBBgwFoAUUWj/kK8CB3U8zNllZGKi
ErhZcjswHQYDVR0OBBYEFLMlpPHQLePNjGxXIQ6AGc9P+aMkMIG1BgNVHREEga0w
gaqCDiouZmFjZWJvb2suY29tgg4qLmZhY2Vib29rLm5ldIILKi5mYmNkbi5uZXSC
CyouZmJzYnguY29tghAqLm0uZmFjZWJvb2suY29tgg8qLm1lc3Nlbmdlci5jb22C
DioueHguZmJjZG4ubmV0gg4qLnh5LmZiY2RuLm5ldIIOKi54ei5mYmNkbi5uZXSC
DGZhY2Vib29rLmNvbYINbWVzc2VuZ2VyLmNvbTA+BgNVHSAENzA1MDMGBmeBDAEC
AjApMCcGCCsGAQUFBwIBFhtodHRwOi8vd3d3LmRpZ2ljZXJ0LmNvbS9DUFMwDgYD
VR0PAQH/BAQDAgOIMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjB1BgNV
HR8EbjBsMDSgMqAwhi5odHRwOi8vY3JsMy5kaWdpY2VydC5jb20vc2hhMi1oYS1z
ZXJ2ZXItZzYuY3JsMDSgMqAwhi5odHRwOi8vY3JsNC5kaWdpY2VydC5jb20vc2hh
Mi1oYS1zZXJ2ZXItZzYuY3JsMIGDBggrBgEFBQcBAQR3MHUwJAYIKwYBBQUHMAGG
GGh0dHA6Ly9vY3NwLmRpZ2ljZXJ0LmNvbTBNBggrBgEFBQcwAoZBaHR0cDovL2Nh
Y2VydHMuZGlnaWNlcnQuY29tL0RpZ2lDZXJ0U0hBMkhpZ2hBc3N1cmFuY2VTZXJ2
ZXJDQS5jcnQwDAYDVR0TAQH/BAIwADCCAX0GCisGAQQB1nkCBAIEggFtBIIBaQFn
AHYA7s3QZNXbGs7FXLedtM0TojKHRny87N7DUUhZRnEftZsAAAGOP4IN3QAABAMA
RzBFAiEArS/xU1Fd7Dol38uDaW3dOnkRwnyOek93f6mmOozeghsCIC9Idpsd0fbM
G7ByW4Yhvu3KYmywL/G6aPsMEnco9mKXAHYASLDja9qmRzQP5WoC+p0w6xxSActW
3SyB2bu/qznYhHMAAAGOP4INSQAABAMARzBFAiEApURec/92Blx7JzufvGS/0ZKh
pySRn+CbihLDbr3EVf4CIGaL3DHm5+XPTIIfvsL/4+VjjzVSTSQfAWgZ7lqCvnuw
AHUA2ra/az+1tiKfm8K7XGvocJFxbLtRhIU0vaQ9MEjX+6sAAAGOP4INMAAABAMA
RjBEAiA1H6SwozkccvBm4SetN3NbTMWCF0c+/lIzhvDVemOHQAIgQYDzV5K4zAWc
8z09NObEnnyP4ZTWYOMTUPP90cAEmzcwDQYJKoZIhvcNAQELBQADggEBAEgR3qwR
X+FT/D/nP+RiDI5KhL7T4NuXQ2PVUn/6VedG4QeyTxQkiK8YPXjLFtKFE/kSEB+v
```

נשמור את ה certeficate השני בקובץ c2.pem

c2.PEM

```
[06/06/2024 14:03] Attacker >>> cat c2.pem
-----BEGIN CERTIFICATE-----
MIIEsTCCA5mgAwIBAgIQBOHnpNxc8vNtwCtCuF0VnzANBgkqhkiG9w0BAQsFADBs
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSswKQYDVQQDEyJEaWdpQ2VydCBIaWdoIEFzc3VyYW5j
ZSBFViBSb290IENBMB4XDTEzMTAyMjEyMDAwMFoXDTI4MTAyMjEyMDAwMFowcDEL
MAkGA1UEBhMCVVMxFTATBgNVBAoTDERpZ2lDZXJ0IEluYzEZMBcGA1UECxMQd3d3
LmRpZ2ljZXJ0LmNvbTEvMC0GA1UEAxMmRGlnaUNlcnQgU0hBMiBIaWdoIEFzc3Vy
YW5jZSBTZXJ2ZXIgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC2
4C/CJAbIbQRf1+8KZAayfSImZRauQkCbztyfn3YHPsMwVYcZuU+UDlqUH1VWtMIC
Kq/QmO4LQNfE0DtyyBSe75CxEamu0si4QzrZCwvV1ZX1QK/IHe1NnF9Xt4ZQaJn1
itrSxwUfqJfJ3KSxgoQtxq2lnMcZgqaFD15EWCo3j/018QsIJzJa9buLnqS9UdAn
4t07QjOjBSjEuyjMmqwrIw14xnvmXnG3Sj4I+4G3FhahnSMSTeXXkgisdaScus0X
sh5ENWV/UyU50RwKmmMbGZJ0aAo3wsJSSMs5WqK24V3B3aAguCGikyZvFEohQcft
bZvySC/zA/WiaJJTL17jAgMBAAGjggFJMIIBRTASBgNVHRMBAf8ECDAGAQH/AgEA
MA4GA1UdDwEB/wQEAwIBhjAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIw
NAYIKwYBBQUHAQEEKDAmMCQGCCsGAQUFBzABhhhodHRwOi8vb2NzcC5kaWdpY2Vy
dC5jb20wSwYDVR0fBEQwQjBAoD6gPIY6aHR0cDovL2NybDQuZGlnaWNlcnQuY29t
L0RpZ2lDZXJ0SGlnaEFzc3VyYW5jZUVWUm9vdENBLmNybDA9BgNVHSAENjA0MDIG
BFUdIAAwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly93d3cuZGlnaWNlcnQuY29tL0NQ
UzAdBgNVHQ4EFgQUUWj/kK8CB3U8zNllZGKiErhZcjswHwYDVR0jBBgwFoAUsT7D
aQP4v0cB1JgmGggC72NkK8MwDQYJKoZIhvcNAQELBQADggEBABiKlYkD5m3fXPwd
aOpKj4PWUS+Na0QWnqxj9dJubISZi6qBcYRb7TROsLd5kinMLYBq8I4g4Xmk/gNH
E+r1hspZcX30BJZr01lYPf7TMSVcGDiEo+afgv2MW5gxTs14nhr9hctJqvIni5ly
/D6q1UEL2tU2ob8cbkdJf17ZSHwD2f2LSaCYJkJA69aSEaRkCldUxPUd1gJea6zu
xICaEnL6VpPX/78whQYwvwt/Tv9XBZ0k7YXDK/umdaisLRbvfXknsuvCnQsH6qqF
0wGjIChBWUMo0oHjqvbsezt3tkBigAVBRQHvFwY+3sAzm2fTYS5yh+Rp/BIAV0Ae
cPUeybQ=
-----END CERTIFICATE-----
```

Step 2: Extract the public key (e, n) from the issuer's certificate.
this is the n value (what coming after Moduls)

נחלץ את ה public key מה certificate , כלומר נמצא את הערך של n

```
[06/07/2024 11:26] Attacker >>> openssl x509 -in c2.pem -noout -mod
ulus
Modulus=B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F7
6073EC330558719B94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C814
9EEF90B111A9AED2C8B8433AD90B0BD5D595F540AFC81DED4D9C5F57B786506899F
58ADAD2C7051FA897C9DCA4B182842DC6ADA59CC71982A6850F5E44582A378FFD35
F10B0827325AF5BB8B9EA4BD51D027E2DD3B4233A30528C4BB28CC9AAC2B230D78C
67BE65E71B74A3E08FB81B71616A19D23124DE5D79208AC75A49CBACD17B21E4435
657F532539D11C0A9A631B199274680A37C2C25248CB395AA2B6E15DC1DDA020B82
1A293266F144A2141C7ED6D9BF2482FF303F5A26892532F5EE3
[06/07/2024 11:26] Attacker >>>
```

נמצא את ה Exponent

```
[06/07/2024 11:33] Attacker >>> openssl x509 -in c2.pem -text -noou
t |grep "Exponent"
                Exponent: 65537 (0x10001)
```

Step 3: Extract the signature from the server's certificate

כעת נריץ את openssl x 509 על c1.pem כדי למצוא את החתימה מה certificate של ה server

```
                                FD:D1:C0:04:9B:37
Signature Algorithm: sha256WithRSAEncryption
     48:11:de:ac:11:5f:e1:53:fc:3f:e7:3f:e4:62:0c:8e:4a:84:
     be:d3:e0:db:97:43:63:d5:52:7f:fa:55:e7:46:e1:07:b2:4f:
     14:24:88:af:18:3d:78:cb:16:d2:85:13:f9:12:10:1f:af:2a:
     73:91:77:e9:45:13:cc:51:2f:89:73:ea:31:bd:fa:b8:52:a9:
     36:ab:ff:b8:87:04:41:87:46:7a:f5:c4:ab:3d:dd:2f:7a:9b:
     64:eb:15:e7:da:cc:c4:f2:f2:5c:e6:ef:8f:ed:4c:16:1e:d0:
     15:0d:e8:ee:57:0b:5e:01:5b:ca:f4:bf:02:1d:b9:9f:c3:c8:
     fb:e6:cd:62:6f:74:bb:8d:af:4d:75:76:a9:bc:ba:1f:35:69:
     fe:6c:f5:09:07:3d:6b:4f:55:53:d9:95:28:a9:00:5f:0b:d4:
     4f:aa:be:5e:a8:83:f3:42:d2:45:bc:37:10:0b:92:cd:17:4b:
     95:7c:ca:54:b2:8d:cf:56:cb:1b:ab:fe:04:37:81:84:01:3f:
     f4:a6:36:5f:30:e2:45:59:d0:73:e1:c2:f5:85:f5:3b:e5:11:
     db:ae:58:47:14:2a:c6:52:43:5e:b6:7d:61:b3:e9:55:30:9d:
     07:7c:32:bd:52:21:bc:1c:83:f1:cc:0a:f3:14:6b:b0:ba:f3:
     0c:56:9d:57
```

Now we will use this: cat signature | tr -d '[:space:]:' on the signature

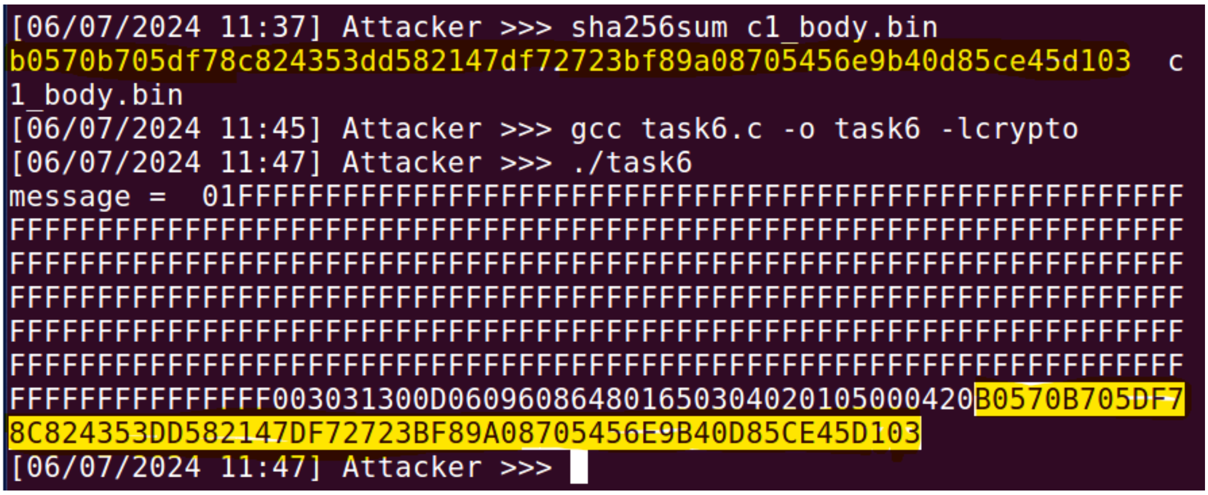
```
[06/06/2024 14:06] Attacker >>> sudo vi signature
[06/06/2024 14:07] Attacker >>> cat signature | tr -d '[:space:]:'
4811deac115fe153fc3fe73fe4620c8e4a84bed3e0db974363d5527ffa55e746e10
7b24f142488af183d78cb16d28513f912101faf2a739177e94513cc512f8973ea31
bdfab852a936abffb887044187467af5c4ab3ddd2f7a9b64eb15e7daccc4f2f25ce
6ef8fed4c161ed0150de8ee570b5e015bcaf4bf021db99fc3c8fbe6cd626f74bb8d
af4d7576a9bcba1f3569fe6cf509073d6b4f5553d99528a9005f0bd44faabe5ea88
3f342d245bc37100b92cd174b957cca54b28dcf56cb1babfe04378184013ff4a636
5f30e24559d073e1c2f585f53be511dbae5847142ac652435eb67d61b3e955309d0
77c32bd5221bc1c83f1cc0af3146bb0baf30c569d57[06/06/2024 14:07] Attac
ker >>>
```

Step 4: Extract the body of the server's certificate.

```
ker >>> openssl asn1parse -i -in c1.pem
    0:d=0  hl=4 l=1687 cons: SEQUENCE
    4:d=1  hl=4 l=1407 cons:  SEQUENCE
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  16 prim:   INTEGER           :0B1058E7787C4E60C
6C5CD0B83C2DA0F
   31:d=2  hl=2 l=  13 cons:   SEQUENCE
   33:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEnc
ryption
   44:d=3  hl=2 l=   0 prim:    NULL
   46:d=2  hl=2 l= 112 cons:   SEQUENCE
   48:d=3  hl=2 l=  11 cons:    SET
   50:d=4  hl=2 l=   9 cons:     SEQUENCE
   52:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   57:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
   61:d=3  hl=2 l=  21 cons:    SET
```

```
[06/06/2024 14:08] Attacker >>> sudo vi signature_final
[06/06/2024 14:09] Attacker >>>  openssl asn1parse -i -in c1.pem -s
trparse 4 -out c1_body.bin -noout
[06/06/2024 14:10] Attacker >>> ls
android                 example
a.out                   example.c
bin                     examples.desktop
c1_body.bin             get-pip.py
c1.pem                  lib
c2.pem                  Music
Customization           Pictures
decrypting_a_message    Public
decrypting_a_message.c  signature
deriving_private_key.c  signature_final
Desktop                 signing_a_message
diriving_private_key    signing_a_message.c
Documents               source
Downloads               Templates
encrypting_a_message    verifing_a_signature
encrypting_a_message.c  verifying_a_signature.c
eran_nissim_native      Videos
[06/06/2024 14:10] Attacker >>> sha256sum c0_body.bin
sha256sum: c0_body.bin: No such file or directory
[06/06/2024 14:11] Attacker >>> sha256sum c1_body.bin
b0570b705df78c824353dd582147df72723bf89a08705456e9b40d85ce45d103  c
1_body.bin
[06/06/2024 14:11] Attacker >>>
```

נריץ את Sha256 sum על c1_body ונשווה את הערך הזה לערך שאנחנו מקבלים מההודעה ב 6task וכפי שניתן לראות הם זהים. הצלחנו (מה שממורקר בצהוב)



```
[06/07/2024 11:37] Attacker >>> sha256sum c1_body.bin
b0570b705df78c824353dd582147df72723bf89a08705456e9b40d85ce45d103  c
1_body.bin
[06/07/2024 11:45] Attacker >>> gcc task6.c -o task6 -lcrypto
[06/07/2024 11:47] Attacker >>> ./task6
message =  01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFF003031300D060960864801650304020105000420B0570B705DF7
8C824353DD582147DF72723BF89A08705456E9B40D85CE45D103
[06/07/2024 11:47] Attacker >>> 
```

קובץ הפייתון שהרצנו של task6.c שמכיל בתוכו את כל הערכים שכתובים למעלה:

```c
#include <stdio.h>
#include <openssl/bn.h>
void printBN(char *msg, BIGNUM * a)
{
/* Use BN_bn2hex(a) for hex string
* Use BN_bn2dec(a) for decimal string */
char * number_str = BN_bn2hex(a);
printf("%s %s\n", msg, number_str);
OPENSSL_free(number_str);
}

int main(){

	BN_CTX *ctx = BN_CTX_new();

	BIGNUM *e = BN_new();
	BIGNUM *M = BN_new();
	BIGNUM *n = BN_new();
	BIGNUM *message = BN_new();
	BIGNUM *S = BN_new();

	BN_hex2bn(&e, "10001");

	BN_hex2bn(&M,"4c61756e63682061206d697373696c652e");
	// This is the value from openssl x509 -in c2.pem -noout -modulus (This is the n)
	BN_hex2bn(&n, "B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC330558719B94F940E5A941F5556B4C2022AA
	               "FD098EE0B40D7C4D03B72C8149EEF90B111A9AED2C8B8433AD90B0BD5D595F540AFC81DED4D9C5F57B7865068
	               "99F58ADAD2C7051FA897C9DCA4B182842DC6ADA59CC71982A6850F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D
	               "027E2DD3B4233A30528C4BB28CC9AAC2B230D78C67BE65E71B74A3E08FB81B71616A19D23124DE5D792
	               "08AC75A49CBACD17B21E4435657F532539D11C0A9A631B199274680A37C2C25248CB395AA2B6E15DC1DDA020B821A293266F144A2141C7ED6D9BF2482FF303F5A26892532F5EE3");

	// This is the value of the signature
	BN_hex2bn(&S, "4811deac115fe153fc3fe73fe4620c8e4a84bed3e0db974363d5527ffa55e746e107b24f142488
	               "af183d78cb16d28513f912101faf2a739177e94513cc512f8973ea31bdfab852a936abffb887044187467af5c4ab3ddd2f7a9b64eb15e7dacc
	               "c4f2f25ce6ef8fed4c161ed0150de8ee570b5e015bcaf4bf021db99fc3c8fbe6cd626f74bb8daf4d7576a9bcba1f3569fe6cf509073d6b4f5553d
	               "99528a9005f0bd44faabe5ea883f342d245bc37100b92cd174b957cca54b28dcf56cb1babfe04378184013ff4a6365f30e24559d073e1c2f585f53b
	               "e511dbae5847142ac652435eb67d61b3e955309d077c32bd5221bc1c83f1cc0af3146bb0baf30c569d57");

	// calculate message = S^e mod n
	BN_mod_exp(message, S, e, n, ctx);

	printBN("message = ", message);

};
```

סיכום:
בתרגיל זה בחרנו את www.facebook.com ונריץ את הפקודה ` openssl s_client -connect www.facebook.com:443 -showcerts`
כדי לקבל את האישורים (1C ו-2C) ולשמור אותם כקבצי PEM.
נחלץ את המפתח הציבורי מהאישור של המנפיק ונמצא את הערכים n וה-Exponent.
לאחר מכן, נשתמש ב-`x509openssl` על 1C כדי למצוא את החתימה מהאישור של השרת ונעבד אותה. נחלץ את גוף האישור של השרת ונריץ עליו SHA-256, נשווה את הערך המתקבל לערך מההודעה ב-6task ונראה שהם זהים.
הקובץ task6.c מכיל את כל הערכים שהוזכרו.

סיכום כללי:
כל אחת מהמטלות הנזכרות מתמקדת ביישום של מערכת RSA לצורך הצפנה ופענוח של הודעות, ובאימות חתימה דיגיטלית על מנת לוודא את תקינותן של ההודעות.

בכל מטלה ישנם שלבים עיקריים:

1. יצירת מפתחות - ניתן לראות שהקוד מתחיל עם יצירת מפתחות ציבוריים ופרטיים, כולל חישובי totient, המפתח הפרטי d, והמפתח הציבורי e ו-n.

2. הצפנה - נעשה שימוש במפתח הציבורי על מנת להצפין את ההודעה, באמצעות חישוב C = (M^e) % n.

3. פענוח- לאחר מכן, ההודעה המצופנת נפענחת באמצעות מפתח הפרטי, באמצעות חישוב M = (C^d) % n.

4. אימות תקינות- בסופו של דבר, ביצירת ובפענוח הודעה, התבצעת אימות תקינות על ידי השוואה בין הודעה מקורית להודעה שנפענחה, ואימת ההצפנה והפענוח.

במקביל, המטלות מדגישות את חוזקן של מערכות הצפנה במפתחות ציבוריים ופרטיים כמו RSA ואת החשיבות של יציבות המפתחות בתהליך ההצפנה והפענוח, וכן באימות תקינות הודעות באמצעות חתימה דיגיטלית.

בסופו של דבר, תוצאות המטלות מדגישות את היכולת של המערכות לבצע הצפנה ופענוח בצורה תקינה ובטוחה, ואת חוזקן של מערכות הצפנה כמו RSA בהגנת מידע ובאימונו של תהליך האימות.

משהו חדשני:

נייצר מפתח פרטי וציבורי בעזרת ספריית rsa בפייתון, נצפין את ההודעה בעזרת המפתח הציבורי ונפענח אותה בעזרת המפתח הפרטי וכפי שניתן לראות לאחר הפענוח שקיבלנו את ההודעה כמו שהייתה לפני כן.

```
import rsa

# Generate key pair
(public_key, private_key) = rsa.newkeys(2048)  # 2048-bit key size (recommended)

# Message to encrypt
message = b"Hello, RSA encryption!"

# Encrypt message with the public key
encrypted_message = rsa.encrypt(message, public_key)

# Decrypt the message using the private key
decrypted_message = rsa.decrypt(encrypted_message, private_key)

# Convert bytes back to string
decrypted_message_str = decrypted_message.decode('utf-8')

# Print keys and decrypted message
print("Public key (n, e):", public_key.n, public_key.e)
print("Private key (n, d):", private_key.n, private_key.d)
print("Original message:", message.decode('utf-8'))
print("Decrypted message:", decrypted_message_str)
```

הפלט של הרצת הקוד:

```
Public key (n, e): 27144490115528234672833871875257855987318399965
23076697343508180737581399580489041087242024708751775688117952936 94
74344456783156092077713118376701544919974648209071484934246635913 92
80134688187254063714207539287832537971243248043012827960159936339 75
87603642150984082079207701676651884710674120557482377914089680309 45
27346378371054546473437344613058042164882043842229100352843543079 22
23356337581143337909760900997400786255396699902499701481429395259 44
13231335604697389617665720284667932797850241945892192618494760899 32
66302106925936186315608633698824040316325478902577496201838164904 01
25294006288175546097549036447188965537
```

```
Private key (n, d): 27144490115528234672833871875257855987318399965
52307669734350818073758139958048904108724202470875177568811795293 69
47434445678315609207771311837670154491997464820907148493424663591 39
28013468818725406371420753928783253797124324804301282796015993633 97
58760364215098408207920770167665188471067412055748237791408968030 94
52734637837105454647343734461305804216488204384222910035284354307 92
22335633758114333790976090099740078625539669990249970148142939525 94
41323133560469738961766572028466793279785024194589219261849476089 93
26630210692593618631560863369882404031632547890257749620183816490 40
12529400628817554609754903644718895 86155643552429279780803141398 97
94740872026432581329347098666825973087751451288293780103592378019 36
50844293341160516973965124971008973309673181059252997657525084376 45
38320255487961595624854772510482775889422707804844246288881796614 39
40261559640978419210014857572466386696787984313860451735957169215 40
61137418414043722334731526588329124063792267651304191381547258791 89
92304461255406664370291593862975488426694350306770810262971450187 74
68574531548823046497415608458321176759077953290377841114946285945 30
25409915996127021132012245726110578758338489400623424573343875997 81
86118385244164444126664066091630506591012379203 3
```

```
Original message: Hello, RSA encryption!
Decrypted message: Hello, RSA encryption!
```