



**Πανεπιστήμιο Δυτικής Αττικής
Σχολή Μηχανικών
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών**

**ΕΡΓΑΣΤΗΡΙΟ ΑΣΦΑΛΕΙΑ ΣΤΗΝ ΤΕΧΝΟΛΟΓΙΑ
ΤΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΦΙΛΙΠΠΟΣ ΠΑΠΑΓΕΩΡΓΙΟΥ - 21390174**

ΗΜΕΡΟΜΗΝΙΑ ΠΑΡΑΔΟΣΗΣ:

Κυριακή 21 Απριλίου 2024 - 11:55 μ.μ.

ΟΜΑΔΑ ΕΡΓΑΣΤΗΡΙΟΥ:

ΑΣΦ05 - ΤΕΤΑΡΤΗ 11:00

Υπευθύνος Ομάδας:

Γεωργούλας Αγγελος

ΕΡΓΑΣΙΑ:ΕΡΓΑΣΙΑ 2(RSA CRYPTOGRAPHY)

Στην παρακάτω εργασία θα δούμε τα εξής:

1. Δημιουργία ιδιωτικού κλειδιού: Θα υπολογίσουμε το modulo N , το $\Phi(N)$, και το ιδιωτικό κλειδί d , δεδομένων των πρώτων αριθμών p , q και του e .
2. Κρυπτογράφηση μηνύματος: Θα κρυπτογραφήσουμε και θα αποκρυπτογραφήσουμε ένα μήνυμα (το ονοματεπώνυμό μας σε μορφή συμβολοσειράς ASCII) με χρήση του δημόσιου και ιδιωτικού κλειδιού που έχουμε δημιουργήσει.
3. Αποκρυπτογράφηση μηνύματος: Θα αποκρυπτογραφήσουμε ένα δοθέν κρυπτογραφημένο μήνυμα c και θα το μετατρέψουμε πίσω σε αναγνώσιμη μορφή ASCII.
4. Υπογραφή μηνύματος: Θα παράγουμε μια ψηφιακή υπογραφή για ένα μήνυμα της επιλογής μας και θα ελέγχουμε τις διαφορές στις υπογραφές μετά από μικρή αλλαγή στο μήνυμα.
5. Επαλήθευση Υπογραφής: Θα ελέγχουμε την εγκυρότητα δύο υπογραφών δεδομένων των δημοσίων κλειδιών των εκδοτών και των μηνυμάτων που έχουν υπογράψει.
6. Μη αυτόματη επαλήθευση πιστοποιητικού X.509: Θα ελέγχουμε την ψηφιακή υπογραφή ενός πιστοποιητικού X.509 με τη χρήση του προγράμματος που αναπτύξαμε, εξάγοντας τα απαραίτητα στοιχεία από το πιστοποιητικό.

Αυτή η εργασία θα μας βοηθήσει να εμβαθύνουμε στην κατανόηση του κρυπτογραφικού αλγορίθμου RSA, της διαδικασίας κρυπτογράφησης και αποκρυπτογράφησης, της δημιουργίας και επαλήθευσης ψηφιακών υπογραφών, καθώς και της επαλήθευσης της υπογραφής πιστοποιητικών X.509, ενισχύοντας τις γνώσεις μας στην εφαρμογή πρακτικών τεχνικών κρυπτογραφίας.

Δραστηριότητα 1: Δημιουργία ιδιωτικού κλειδιού

Στην πρώτη δραστηριότητα της εργασίας, θα εξοικειωθούμε με τη διαδικασία δημιουργίας ενός ιδιωτικού κλειδιού χρησιμοποιώντας τον κρυπτογραφικό αλγόριθμο RSA. Αρχικά, θα πρέπει να ορίσουμε δύο πρώτους αριθμούς, p και q , και έναν αριθμό e , ο οποίος θα είναι σχετικά πρώτος με το $\Phi(N)$. Οι δεκαεξαδικές τιμές των p , q , και e θα μας δοθούν ως εξής:

- $p = \text{F7E75FDC469067FFDC4E847C51F452DF}$

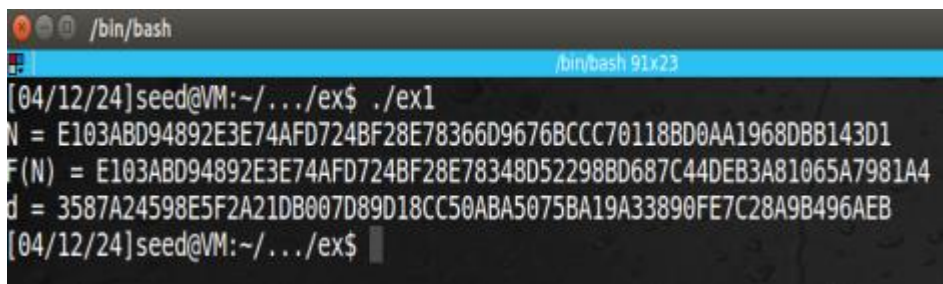
- $q = \text{E85CED54AF57E53E092113E62F436F4F}$

- $e = \text{0D88C3}$

Στη συνέχεια, θα προγραμματίσουμε σε γλώσσα C για να υπολογίσουμε:

1. Το modulo N , το οποίο προκύπτει από το γινόμενο των p και q .
2. Τον αριθμό $\Phi(N)$, ο οποίος υπολογίζεται από τον τύπο $\Phi(N) = (p-1) * (q-1)$.
3. Το ιδιωτικό κλειδί d , το οποίο υπολογίζεται ώστε $d * e \bmod \Phi(N) = 1$.

Αποτέλεσμα Κώδικα.



```
/bin/bash
/bin/bash 91x23
[04/12/24]seed@VM:~/.../ex$ ./ex1
N = E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
F(N) = E103ABD94892E3E74AFD724BF28E78348D52298BD687C44DEB3A81065A7981A4
d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[04/12/24]seed@VM:~/.../ex$
```

Κώδικας

```
1 #include <stdio.h>
2 #include <openssl/bn.h>
3 //21390174 FILIPPOS PAPAGEORGIOU
4 void printBN(char* msg, BIGNUM* a) {
5     char* number_str = BN_bn2hex(a);
6     printf("%s%s\n", msg, number_str);
7     OPENSSL_free(number_str);
8 }
9
10 int main() {
11     BN_CTX *ctx = BN_CTX_new();
12
13     BIGNUM *p = BN_new();
14     BIGNUM *q = BN_new();
15     BIGNUM *e = BN_new();
16     BIGNUM *N = BN_new();
17     BIGNUM *phi = BN_new();
18     BIGNUM *d = BN_new();
19     BIGNUM *p_1 = BN_new();
20     BIGNUM *q_1 = BN_new();
21     BIGNUM *one = BN_new();
22
23     //Ftiaxno p, q, and e
24     BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
25     BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
26     BN_hex2bn(&e, "0D88C3");
27
28     // Ypologizo N = p * q
29     BN_mul(N, p, q, ctx);
30
31     // Ypologizo F(n) = (p-1) * (q-1)
32     BN_hex2bn(&one, "1");
33     BN_sub(p_1, p, one);
34     BN_sub(q_1, q, one);
35     BN_mul(phi, p_1, q_1, ctx);
36
37     // Ypologizo d opu to d * e mod phi = 1
38     BN_mod_inverse(d, e, phi, ctx);
39
40     // Print
41     printBN("N = ", N);
42     printBN("F(N) = ", phi);
43     printBN("d = ", d);
44
45     BN_free(p);
46     BN_free(q);
47     BN_free(e);
48     BN_free(N);
49     BN_free(phi);
50     BN_free(d);
51     BN_free(p_1);
52     BN_free(q_1);
53     BN_free(one);
54     BN_CTX_free(ctx);
55
56     return 0;
57 }
```

Δραστηριότητα 2: ΚΡΥΠΤΟΓΡΑΦΗΣΗ ΜΥΝΗΜΑΤΟΣ

Στη δεύτερη δραστηριότητα της εργασίας μας, θα εξασκηθούμε στην κρυπτογράφηση και αποκρυπτογράφηση ενός απλού μηνύματος με χρήση του κρυπτογραφικού αλγορίθμου RSA. Θα χρησιμοποιήσουμε το δημόσιο και ιδιωτικό κλειδί που δημιουργήσαμε στην πρώτη δραστηριότητα. Θα γράψουμε ένα πρόγραμμα σε γλώσσα C για την κρυπτογράφηση και αποκρυπτογράφηση του μηνύματος "Όνομα Επωνυμο".

1. Μετατροπή του Μηνύματος σε Δεκαεξαδική Μορφή:

Για να κρυπτογραφήσουμε το μήνυμα, πρέπει πρώτα να το μετατρέψουμε από ASCII σε δεκαεξαδική μορφή. Μπορούμε να το κάνουμε αυτό στο terminal χρησιμοποιώντας Python:

2. Κρυπτογράφηση του Μηνύματος:

Χρησιμοποιούμε το δημόσιο κλειδί (e, N) για να κρυπτογραφήσουμε το μήνυμα.

3. Αποκρυπτογράφηση του Μηνύματος:

Με το ιδιωτικό κλειδί (d) που βρήκαμε στην προηγούμενη δραστηριότητα, θα αποκρυπτογραφήσουμε το κρυπτογραφημένο μήνυμα.

4. Εκτύπωση Αποτελεσμάτων:

Θα πρέπει να εκτυπώσουμε και το αρχικό και το αποκρυπτογραφημένο μήνυμα για επαλήθευση. Επίσης, θα καταγράψουμε την εκτέλεση των παραπάνω στο terminal για να περιληφθούν στην τελική εργασία.

```
print("Filippos Papageorgiou".encode("utf-8").hex())
```

```
46696c6970706f73205061706167656f7267696f75
```

```
print(bytes.fromhex("46696c6970706f73205061706167656f7267696f75").decode("utf-8"))
```

```
Filippos Papageorgiou
```

```

1  #include <stdio.h>
2  #include <openssl/bn.h>
3  int main() {
4      BN_CTX *ctx = BN_CTX_new();
5      BIGNUM *p = BN_new();
6      BIGNUM *q = BN_new();
7      BIGNUM *N = BN_new();
8      BIGNUM *e = BN_new();
9      BIGNUM *d = BN_new();
10     BIGNUM *phi = BN_new();
11     BIGNUM *msg = BN_new();
12     BIGNUM *enc_msg = BN_new();
13     BIGNUM *dec_msg = BN_new();
14     char *enc_str;
15     char *dec_str;
16
17
18     BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
19     BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
20     BN_hex2bn(&e, "0D88C3");
21
22
23     BN_mul(N, p, q, ctx);
24
25
26     BIGNUM *p1 = BN_new();
27     BIGNUM *q1 = BN_new();
28     BN_sub(p1, p, BN_value_one()); // p-1
29     BN_sub(q1, q, BN_value_one()); // q-1
30     BN_mul(phi, p1, q1, ctx);
31
32
33     BN_mod_inverse(d, e, phi, ctx);
34
35     //Dilosi hex
36     BN_hex2bn(&msg, "46696c6970706f73205061706167656f7267696f75");
37
38     // Encrypt to minima: enc_msg = msg^e mod N
39     BN_mod_exp(enc_msg, msg, e, N, ctx);
40     enc_str = BN_bn2hex(enc_msg);
41     printf("Encrypted Message: %s\n", enc_str);
42     OPENSSL_free(enc_str);
43
44     // Decrypt to minima: dec_msg = enc_msg^d mod N
45     BN_mod_exp(dec_msg, enc_msg, d, N, ctx);
46     dec_str = BN_bn2hex(dec_msg);
47     printf("Decrypted Message: %s\n", dec_str);
48     OPENSSL_free(dec_str);
49
50
51     // Cleanup
52     BN_free(p);
53     BN_free(q);
54     BN_free(N);
55     BN_free(e);
56     BN_free(d);
57     BN_free(phi);
58     BN_free(p1);
59     BN_free(q1);

```

Αποτέλεσμα Κώδικα.

```

04/12/24]seed@VM:~/.../ex$ gcc -o ex ex.c -lcrypto
04/12/24]seed@VM:~/.../ex$ ./ex
Encrypted Message: C2628AA18B8F73B387059F6788497582DBCE56B9C2D1453A863844DE16C1F7F5
Decrypted Message: 46696c6970706f73205061706167656f7267696f75
04/12/24]seed@VM:~/.../ex$

```

Το οποίο είναι το κρυπτογραφημένο μήνυμα

Δραστηριότητα 3: Αποκρυπτογράφηση μηνύματος

Στην τρίτη δραστηριότητα της εργασίας, θα εστιάσουμε στην αποκρυπτογράφηση ενός δοθέντος κρυπτογραφημένου μηνύματος (c), χρησιμοποιώντας τα ήδη υπολογισμένα δημόσιο και ιδιωτικό κλειδί από την πρώτη δραστηριότητα. Αυτή η διαδικασία θα γίνει με τη χρήση ενός προγράμματος σε γλώσσα C, και το αποκρυπτογραφημένο μήνυμα θα πρέπει να μετατραπεί πίσω σε ASCII μορφή για να γίνει αναγνώσιμο.

Για να κάνω αυτήν την δραστηριότητα δήλωσα κατευθείαν σαν encrypt το μήνυμα που μας δοθήκε και χρησιμοποιήσα τον ίδιο κωδικά με την προηγούμενη ασκήση.

```
//Dilosi hex
BN_hex2bn(&msg, "9D368A5F8F562EB9553F6081B492C3D9527298DA5557B1895E6F7B7C8B01A308");

/*// Encrypt to minima: enc_msg = msg^e mod N
BN_mod_exp(enc_msg, msg, e, N, ctx);
enc_str = BN_bn2hex(enc_msg);
printf("Encrypted Message: %s\n", enc_str);
OPENSSL_free(enc_str);*/

// Decrypt to minima: dec_msg = enc_msg^d mod N
BN_mod_exp(dec_msg, enc_msg, d, N, ctx);
dec_str=BN_bn2hex(dec_msg);
printf("Decrypted Message: %s\n", dec_str);
OPENSSL_free(dec_str);
```

```
[04/12/24]seed@VM:~/.../ex$ ./ex
Decrypted Message: 496E666F53656320537072696E672053656D65737465722032303234
[04/12/24]seed@VM:~/.../ex$
```

```
print(bytes.fromhex("496E666F53656320537072696E672053656D65737465722032303234").decode("utf-8"))
```

InfoSec Spring Semester 2024

Δραστηριότητα 4: Υπογραφή μηνύματος

Στην τέταρτη δραστηριότητα, θα αναπτύξουμε ένα πρόγραμμα για την παραγωγή ψηφιακών υπογραφών ενός μηνύματος, χρησιμοποιώντας τον RSA αλγόριθμο με τα δημόσια και ιδιωτικά κλειδιά που έχουμε ήδη δημιουργήσει. Θα παραχθούν δύο υπογραφές: μία για το αρχικό μήνυμα και μία για ένα ελαφρώς τροποποιημένο μήνυμα, με σκοπό να διερευνήσουμε τις διαφορές μεταξύ των δύο.

- Προετοιμασία του μηνύματος

Πρώτα, θα πρέπει να μετατρέψουμε το μήνυμα από ASCII σε δεκαεξαδική μορφή για να μπορέσουμε να το διαχειριστούμε με τον RSA αλγόριθμο. Αυτό μπορεί να γίνει είτε απευθείας μέσα στον κώδικα, είτε χρησιμοποιώντας εντολές Python.

```
print("Univercity of west Attica".encode("utf-8").hex())
print(bytes.fromhex("556e6976657263697479206f666207765737420417474696361").decode("utf-8"))
```

556e6976657263697479206f666207765737420417474696361
Univercity of west Attica

```
1  #include <stdio.h>
2  #include <openssl/bn.h>
3
4  int main() {
5      BN_CTX *ctx = BN_CTX_new();
6      BIGNUM *p = BN_new();
7      BIGNUM *q = BN_new();
8      BIGNUM *N = BN_new();
9      BIGNUM *e = BN_new();
10     BIGNUM *d = BN_new();
11     BIGNUM *phi = BN_new();
12     BIGNUM *msg = BN_new();
13     BIGNUM *sig = BN_new();
14     char *sig1_str;
15     char *sig2_str;
16
17
18     BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
19     BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
20     BN_hex2bn(&e, "0D88C3");
21
22     BN_mul(N, p, q, ctx);
23
24     BIGNUM *p1 = BN_new();
25     BIGNUM *q1 = BN_new();
26     BN_sub(p1, p, BN_value_one()); // p-1
27     BN_sub(q1, q, BN_value_one()); // q-1
28     BN_mul(phi, p1, q1, ctx);
29
30
31     BN_mod_inverse(d, e, phi, ctx);
32
33     //Dilos1 hex
34     //University of West attica
35     BN_hex2bn(&msg, "556e6976657263697479206f666207765737420417474696361");
36
37
38     BN_mod_exp(sig, msg, d, N, ctx);
39     sig1_str = BN_bn2hex(sig);
40     printf("Signature Original Message: %s\n", sig1_str);
41     OPENSSL_free(sig1_str);
42
43     //Allagi minimatos
44     BN_hex2bn(&msg, "5556e6976657263697479206f6662077657374204174746963611");
45     BN_mod_exp(sig, msg, d, N, ctx);
46     sig2_str = BN_bn2hex(sig);
47     printf("Signature Original Message: %s\n", sig2_str);
48     OPENSSL_free(sig2_str);
49
50
51
52     // Cleanup
53     BN_free(p);
54     BN_free(q);
55     BN_free(N);
56     BN_free(e);
57     BN_free(d);
58     BN_free(phi);
59 }
```



```
Signature Original Message: 6A743F41FAA3CA6698EB5283CF700748A94A59EF807B6205957685AF39  
0A49A  
Signature Original Message: B89613D3F6313ED0CA3871BDD07E218DF3EE031475025046C14C3B10F4  
4796A  
[04/12/24]seed@VM:~/.../ex$
```

Όταν υπογράψουμε ένα μήνυμα και το τροποποιήσουμε έστω και ελάχιστα, η παραγόμενη υπογραφή θα διαφέρει δραματικά. Αυτό δείχνει την ευαισθησία των κρυπτογραφικών αλγορίθμων σε αλλαγές στα δεδομένα, καθώς ακόμα και η παραμικρή διαφορά στο αρχικό μήνυμα προκαλεί σημαντική αλλαγή στην τελική υπογραφή, επιβεβαιώνοντας την ακεραιότητα του μηνύματος.

Δραστηριότητα 5: Επαλήθευση Υπογραφής

Για την πέμπτη δραστηριότητα, θα αναπτύξουμε ένα πρόγραμμα σε γλώσσα C που θα επιβεβαιώνει την αυθεντικότητα των ψηφιακών υπογραφών χρησιμοποιώντας τον αλγόριθμο RSA. Στην περίπτωση αυτή, η επιβεβαίωση γίνεται με την αποκρυπτογράφηση της υπογραφής με το δημόσιο κλειδί και τη σύγκριση του αποτελέσματος με το αρχικό μήνυμα.

```

1  #include <stdio.h>
2  #include <openssl/bn.h>
3
4  void printBN(char *msg, BIGNUM *a) {
5      char *number_str = BN_bn2dec(a);
6      printf("%s %s\n", msg, number_str);
7      OPENSSL_free(number_str);
8  }
9
10 int main() {
11     BN_CTX *ctx = BN_CTX_new();
12     BIGNUM *N = BN_new();
13     BIGNUM *e = BN_new();
14     BIGNUM *s = BN_new();
15     BIGNUM *m_decrypted = BN_new();
16     BIGNUM *m_original = BN_new();
17     BIGNUM *tampered_s = BN_new();
18
19     // Case 1
20     printf("Case A:\n");
21     BN_hex2bn(&N, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
22     BN_hex2bn(&e, "010001"); // 65537 to dec
23     BN_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
24     BN_mod_exp(m_decrypted, s, e, N, ctx);
25     BN_hex2bn(&m_original, "4c61756e63682061206d697373696c652e"); // "Launch a missile."
26     printBN("Encrypted msg: ", m_decrypted);
27     printBN("Original msg: ", m_original);
28
29     if (BN_cmp(m_decrypted, m_original) == 0)
30         printf("sig is correct.\n");
31     else
32         printf("sig is not correct.\n");
33
34     printf("\nCase A Forged Signature\n");
35     BN_hex2bn(&tampered_s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");
36     BN_mod_exp(m_decrypted, tampered_s, e, N, ctx);
37     printBN("Encrypted msg: ", m_decrypted);
38     if (BN_cmp(m_decrypted, m_original) == 0)
39         printf("sig is correct.\n");
40     else
41         printf("sig is not correct.\n");
42
43     printf("\nCase B:\n");
44     BN_hex2bn(&N, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDD3A4D0CB81629242FB1A5");
45     BN_hex2bn(&e, "010001");
46     BN_hex2bn(&s, "DB3F7CDB93483FC1E70E4EACA650E3C6505A3E5F49EA6EDF3E95E9A7C6C7A320");
47     BN_mod_exp(m_decrypted, s, e, N, ctx);
48     BN_hex2bn(&m_original, "506c65617365207472616e73666572206d6520243230302e416c6963652e");
49     printBN("Encrypted msg: ", m_decrypted);
50
51     printBN("Original msg: ", m_original);
52     if (BN_cmp(m_decrypted, m_original) == 0)
53         printf("sig is correct.\n");
54     else
55         printf("sig is not correct.\n");
56
57     BN_free(N);

```

```

[04/12/24]seed@VM:~/.../ex$ ./ex5
Case A:
Encrypted msg: 25991004739255778713631969737248063907118
Original msg: 25991004739255778713631969737248063907118
sig is correct.

Case A Forged Signature
Encrypted msg: 65710982802337676312353566637294969595512391135201560281913984954792299
643540
sig is not correct.

Case B:
Encrypted msg: 14209588700510056180835069687111635766955028506812121070001372497218603
5502
Original msg: 142095887005100561808350696871116357669550285068121210700013724972186035
502
sig is correct.
[04/12/24]seed@VM:~/.../ex$

```

Δραστηριότητα 6: Μη αυτόματη επαλήθευση πιστοποιητικού X.509

Για τη Δραστηριότητα 6, θα εφαρμόσουμε τα βήματα για να επαληθεύσουμε την υπογραφή ενός πιστοποιητικού X.509 που λάβαμε από έναν web server. Ας δούμε πώς θα προχωρήσουμε με την υλοποίηση:

Βήμα 1: Λήψη του Πιστοποιητικού X.509

Για να λάβουμε το πιστοποιητικό, μπορούμε να χρησιμοποιήσουμε την εντολή **openssl** στο terminal. Θα συνδεθούμε σε έναν πραγματικό server και θα καταγράψουμε το πιστοποιητικό, ετώ διαλέξα ως site το open.spotify.com

```
04/12/24]seed@VM:~/.../ex$ openssl s_client -connect open.spotify.com:443 -showcerts
```

Αυτή η εντολή θα συνδεθεί στον server της Spotify στην πόρτα 443, που είναι η προεπιλεγμένη για HTTPS, και θα εμφανίσει τα πιστοποιητικά που χρησιμοποιεί ο server. Η έξοδος της εντολής θα περιλαμβάνει το πλήρες chain των πιστοποιητικών από το πιστοποιητικό του server μέχρι το root πιστοποιητικό. Και φτιάχνω τα δύο .pem αρχεία.

Βήμα 2: Εξαγάγετε το δημόσιο κλειδί (e, N) από το πιστοποιητικό του εκδότη.

Το Openssl παρέχει εντολές για την εξαγωγή ορισμένων χαρακτηριστικών από τα πιστοποιητικά x509. Μπορούμε να εξαγάγουμε την τιμή του n χρησιμοποιώντας την επιλογή -modulus.

```
04/12/24]seed@VM:~/.../ex$ openssl x509 -in c1.pem -text -noout
```

Αυτή η εντολή θα εκτυπώσει τα στοιχεία του πιστοποιητικού, συμπεριλαμβανομένου του δημόσιου κλειδιού. Συνήθως, το δημόσιο κλειδί e είναι το 65537 (0x10001 σε δεκαεξαδικό), και το N θα βρίσκεται στην έξοδο της εντολής.

```

Public Key Algorithm: rsaEncryption
Public-Key: (2048 bit)
Modulus:
  00:cc:f7:10:62:4f:a6:bb:63:6f:ed:90:52:56:c5:
  6d:27:7b:7a:12:56:8a:f1:f4:f9:d6:e7:e1:8f:bd:
  95:ab:f2:60:41:15:70:db:12:00:fa:27:0a:b5:57:
  38:5b:7d:b2:51:93:71:95:0e:6a:41:94:5b:35:1b:
  fa:7b:fa:bb:c5:be:24:30:fe:56:ef:c4:f3:7d:97:
  e3:14:f5:14:4d:cb:a7:10:f2:16:ea:ab:22:f0:31:
  22:11:61:69:90:26:ba:78:d9:97:1f:e3:7d:66:ab:
  75:44:95:73:c8:ac:ff:ef:5d:0a:8a:59:43:e1:ac:
  b2:3a:0f:f3:48:fc:d7:6b:37:c1:63:dc:de:46:d6:
  db:45:fe:7d:23:fd:90:e8:51:07:1e:51:a3:5f:ed:
  49:46:54:7f:2c:88:c5:f4:13:9c:97:15:3c:03:e8:
  a1:39:dc:69:0c:32:c1:af:16:57:4c:94:47:42:7c:
  a2:c8:9c:7d:e6:d4:4d:54:af:42:99:a8:c1:04:c2:
  77:9c:d6:48:e4:ce:11:e0:2a:80:99:f0:43:70:cf:
  3f:76:6b:d1:4c:49:ab:24:5e:c2:0d:82:fd:46:a8:
  ab:6c:93:cc:62:52:42:75:92:f8:9a:fa:5e:5e:b2:
  b0:61:e5:1f:1f:b9:7f:09:98:e8:3d:fa:83:7f:47:
  69:a1
Exponent: 65537 (0x10001)

```

Βήμα 3: Εξάγετε την ψηφιακή υπογραφή από το πιστοποιητικό του server.

Δεν υπάρχει συγκεκριμένη εντολή openssl για την εξαγωγή του πεδίου υπογραφής. Ωστόσο, μπορούμε να εκτυπώσουμε όλα τα πεδία και στη συνέχεια να αντιγράψουμε και να επικολλήσουμε το block υπογραφής σε ένα αρχείο (Σημείωση: αν ο αλγόριθμος υπογραφής που χρησιμοποιείται στο πιστοποιητικό δεν βασίζεται στον RSA, θα πρέπει να βρείτε ένα άλλο πιστοποιητικό).

```
[04/12/24]seed@VM:~/.../ex$ openssl x509 -in c0.pem -text -noout
```

```

Signature Algorithm: sha256WithRSAEncryption
 65:37:c4:be:93:ef:e8:db:d4:4a:b3:9a:79:73:5c:39:7c:10:
 a7:db:24:56:8d:4a:81:9e:96:a8:40:0d:0a:9f:01:83:ae:10:
 70:bf:13:ba:b8:03:69:dd:cd:9f:18:7d:40:c3:57:4b:68:7b:
 a7:b6:07:62:7a:ae:be:0c:c4:c4:26:3d:ad:de:fc:c9:3c:cf:
 d9:d4:80:f8:04:cf:31:d5:9a:33:54:10:0f:75:ea:1a:ca:88:
 bd:89:c0:68:23:f9:28:13:44:b2:d0:5a:4e:f8:da:ab:b9:a6:
 f0:df:94:0c:0c:37:24:88:b9:f2:51:ed:f1:82:4f:5c:e3:f3:
 47:2b:b4:cb:80:54:f6:5e:ca:6c:3c:0d:ba:c8:24:f4:22:62:
 1b:c6:18:e0:00:48:f9:43:57:6b:dd:e5:f3:7d:39:ae:94:97:
 52:c5:00:ac:e2:8f:9f:e7:5f:cc:f3:33:c2:53:e0:a3:8e:39:
 51:ad:f1:3b:1b:92:1e:b3:39:7b:b1:af:30:85:3e:bc:89:8e:
 9b:42:2e:4b:86:94:b6:92:1f:52:18:75:53:df:57:ed:4b:71:
 03:bc:e1:40:c2:65:52:90:4d:21:f0:db:ce:55:9c:18:57:5e:
 c9:5f:7d:b8:79:ca:e8:d5:25:96:6f:43:dc:be:2b:68:08:66:
 3d:79:5f:8b

```



```
[04/12/24]seed@VM:~/.../ex$ subl signature
[04/12/24]seed@VM:~/.../ex$ cat signature | tr -d '[:space:]'
6537c4be93efe8dbd44ab39a79735c397c10a7db24568d4a819e96a8400d0a9f0183ae1070bf13bab80369ddcd9f187d40c3574b687
a7b607627aaebe0cc4c4263daddefcc93ccfd9d480f804cf31d59a3354100f75ea1aca88bd89c06823f9281344b2d05a4ef8daabb9a
f0df940c0c372488b9f251edf1824f5ce3f3472bb4cb8054f65eca6c3c0dbac824f422621bc618e00048f943576bdde5f37d39ae949
52c500ace28f9fe75fccf333c253e0a38e3951adf13b1b921eb3397bb1af30853ebc898e9b422e4b8694b6921f52187553df57ed4b7
03bce140c26552904d21f0dbce559c18575ec95f7db879cae8d525966f43dcbe2b6808663d795f8b[04/12/24]seed@VM:~/.../ex$
```

Βήμα 4:

Εξαγάγετε το σώμα του πιστοποιητικού του server. Μια Αρχή Πιστοποίησης (CA) δημιουργεί την υπογραφή για ένα πιστοποιητικό server, αρχικά υπολογίζοντας το hash του πιστοποιητικού και στη συνέχεια υπογράφοντας το hash. Για να επαληθεύσουμε την υπογραφή, πρέπει επίσης να δημιουργήσουμε το hash από ένα πιστοποιητικό. Δεδομένου ότι ο κατακερματισμός δημιουργείται πριν από τον υπολογισμό της υπογραφής, πρέπει να αποκλείσουμε το block υπογραφής ενός πιστοποιητικού κατά τον υπολογισμό του hash. Η εύρεση του τμήματος του πιστοποιητικού που χρησιμοποιείται για τη δημιουργία του hash είναι αρκετά δύσκολη χωρίς την καλή κατανόηση της μορφής του πιστοποιητικού. 14 Τα πιστοποιητικά X.509 κωδικοποιούνται σύμφωνα με το πρότυπο ASN.1 (Abstract Syntax Notation.One), οπότε αν μπορούσαμε να αναλύσουμε τη δομή ASN.1, μπορούμε εύκολα να εξάγουμε οποιοδήποτε πεδίο από ένα πιστοποιητικό. Το Openssl έχει μια εντολή που ονομάζεται `asn1parse`, η οποία μπορεί να χρησιμοποιηθεί για την ανάλυση ενός πιστοποιητικού X.509.

```
[04/12/24]seed@VM:~/.../ex$ openssl asn1parse -i -in c0.pem
0:d=0 hl=4 l=1736 cons: SEQUENCE
4:d=1 hl=4 l=1456 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 :0AD472FA89D77BDC663EF48B0110EC2E
31:d=2 hl=2 l= 13 cons: SEQUENCE
33:d=3 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption
44:d=3 hl=2 l= 0 prim: NULL
46:d=2 hl=2 l= 89 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
63:d=4 hl=2 l= 19 cons: SEQUENCE
65:d=5 hl=2 l= 3 prim: OBJECT :organizationName
70:d=5 hl=2 l= 12 prim: PRINTABLESTRING :DigiCert Inc
84:d=3 hl=2 l= 51 cons: SET
86:d=4 hl=2 l= 49 cons: SEQUENCE
88:d=5 hl=2 l= 3 prim: OBJECT :commonName
93:d=5 hl=2 l= 42 prim: PRINTABLESTRING :DigiCert Global G2 TLS RSA SHA256 202
137:d=2 hl=2 l= 30 cons: SEQUENCE
139:d=3 hl=2 l= 13 prim: UTCTIME :240129000000Z
154:d=3 hl=2 l= 13 prim: UTCTIME :250128235959Z
169:d=2 hl=2 l= 78 cons: SEQUENCE
171:d=3 hl=2 l= 11 cons: SET
173:d=4 hl=2 l= 9 cons: SEQUENCE
175:d=5 hl=2 l= 3 prim: OBJECT :countryName
180:d=5 hl=2 l= 2 prim: PRINTABLESTRING :SE
184:d=3 hl=2 l= 18 cons: SET
186:d=4 hl=2 l= 16 cons: SEQUENCE
188:d=5 hl=2 l= 3 prim: OBJECT :localityName
```

```
[04/12/24]seed@VM:~/.../ex$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
[04/12/24]seed@VM:~/.../ex$ sha256sum c0_body.bin
48579ee4e7d2b5b64d1ed0a7b6e8579b85b070becdc4b9e6b08edf5b1431823f c0_body.bin
[04/12/24]seed@VM:~/.../ex$
```

Βήμα 5:

Επαληθεύστε την υπογραφή. Τώρα έχουμε όλες τις πληροφορίες, συμπεριλαμβανομένου του δημόσιου κλειδιού της CA, της υπογραφής της CA και του σώματος του πιστοποιητικού του server. Θα πρέπει να εκτελέσετε το δικό σας πρόγραμμα για να επαληθεύσετε αν η υπογραφή είναι έγκυρη ή όχι.

```
[04/12/24]seed@VM:~/.../ex$ openssl dgst -sha256 -verify issuer_pubkey.pem -signature signature.bin c0_body
bin
Verification Failure
```