



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

ΑΣΦΑΛΕΙΑ ΣΤΗΝ ΤΕΧΝΟΛΟΓΙΑ
ΤΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
(Εργαστήριο)

Cryptography
RSA

ΑΣΦ03 - Εργασία 2 (Cryptography)

Ασύμμετρη κρυπτογράφηση,
αποκρυπτογράφηση, ψηφιακή υπογραφή,
hashes, πιστοποιητικά.

Παπαντωνάκης Σταυρός ΑΜ:cse45227

3/5/2020

RSA-Cryptography

Ασύμμετρη Κρυπτογράφηση

Παπαντωνάκης Σταύρος
CSE45227

Αναφορά δεύτερου εργαστηριού ασφάλειας
ΑΣΦ03



Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών
Πανεπιστήμιο Δυτικής Αττικής
Αθηνά
03/05/2020

Copyright ©2020 Παπαντώνακης Σταύρος
Το Παρόν Έργο παρέχεται υπό τους όρους της Άδειας:



Αναφορά Δημιουργού-Μη Εμπορική Χρήση-Παρόμοια Διανομή 4.0 Διεθνής

Το πλήρες κείμενο αυτής της άδειας είναι διαθέσιμο εδώ:
<http://creativecommons.org/licenses/by-nc-sa/4.0/>

Είστε ελεύθερος να:

Διαμοιραστείτε – να αντιγράψετε και αναδιανείμετε το υλικό με οποιοδήποτε μέσο και μορφή.

Προσαρμόσετε – να αναμείξετε, μετασχηματίσετε και να επεκτείνετε το υλικό.

Ο αδειοδότης δεν μπορεί να σας αφαιρέσει αυτές τις ελευθερίες όσο ακολουθείτε τους όρους παρούσας άδειας.

Τύπο τους ακόλουθους όρους:



Αναφορά Δημιουργού – Θα πρέπει να αναφέρετε τον δημιουργό του έργου, να παρέχετε σύνδεσμο προς αυτή την άδεια, και να υποδείξετε τυχόν αλλαγές. Μπορείτε να το κάνετε με οποιοδήποτε εύλογο μέσο, αλλά όχι με τρόπο που να υπονοεί ότι ο αδειοδότης επικροτεί εσάς ή τη χρήση του έργου από εσάς.



Μη Εμπορική Χρήση – Δεν μπορείτε να χρησιμοποιήσετε το υλικό για εμπορικούς σκοπούς.



Παρόμοια Διανομή – Αν αναμείξετε, μετασχηματίσετε ή επεκτείνετε το υλικό, θα πρέπει να διανείμετε τις αλλαγές σας υπό την ίδια άδεια με το πρωτότυπο έργο.

Όχι επιπλέον περιορισμοί – Δεν μπορείτε να εφαρμόσετε νομικούς όρους ή τεχνικά μέσα που να περιορίζουν νομικά τους άλλους για πράξουν σύμφωνα με τις ελευθερίες αυτής της άδειας.

Σημειώσεις:

Δεν χρειάζεται να ακολουθήσετε την άδεια για τμήματα του υλικού που θεωρούνται δημόσια γνώση (public domain) ή όπου η χρήση τους επιτρέπεται εξαιτίας μιας εξαίρεσης ή περιορισμού.

Δεν δίνονται εγγυήσεις. Η άδεια ίσως να μη σας δίνει όλα τα δικαιώματα για την επιδιωκόμενη χρήση. Για παράδειγμα, επιπλέον δικαιώματα όπως δημοσιότητα, ιδιωτικότητα, ή ηθικά δικαιώματα μπορεί να επιβάλλουν περιορισμούς στη χρήση του υλικού.

Το παρόν έργο στοιχειοθετήθηκε σε X_{PLAT}EX. Ο πηγαίος κώδικας του είναι διαθέσιμος στην παρακάτω τοποθεσία:

<https://github.com/lardianos/Security>

Περιεχόμενα

2 Γενικό Πλαίσιο	4
2.1 BIGNUM APIs	5
2.2 Παράδειγμα	6
3 Δραστηριότητες Εργαστηρίου	7
3.1 Δραστηριότητα 1: Δημιουργώντας το ιδιωτικό κλειδί	8
3.2 Δραστηριότητα 2: Κρυπτογράφηση μηνύματος	10
3.3 Δραστηριότητα 3: Αποκρυπτογράφηση μηνύματος	13
3.4 Δραστηριότητα 4: Ύπογραφή μηνύματος	15
3.5 Δραστηριότητα 5: Επαλήθευση Ύπογραφής	17
3.6 Δραστηριότητα 6: Μη αυτόματη επαλήθευση πιστοποιητικού X.509	19

Εισαγωγή

Ο κρυπτογραφικός αλγόριθμος Rivest-Shamir-Adelman (RSA) είναι ένα κρυπτοσύστημα δημοσίου κλειδιού. Είναι ένας ασύμμετρος κρυπτογραφικός αλγόριθμος όπου κάθε χρήστης διαθέτει δύο κλειδιά, ένα ιδιωτικό και ένα δημόσιο. Χρησιμοποιεί στοιχεία από τη θεωρία των αριθμών και σε συνδυασμό με τα ιδιαίτερα μεγάλου μεγέθους κλειδιά επιτυγχάνει κρυπτογράφηση σε αριθμητική υπολογίση που καθιστά αδύνατη την αποκρυπτογράφηση με παραγοντοποίηση. Θεωρείται μέχρι σήμερα ένας ασφαλής κρυπτογραφικός αλγόριθμος.

2 Γενικό Πλαίσιο

Ο αλγόριθμος RSA περιλαμβάνει υπολογισμούς σε μεγάλους αριθμούς. Αυτοί οι υπολογισμοί δεν μπορούν να διεξαχθούν απευθείας στη μηχανή χρησιμοποιώντας απλούς αριθμητικούς τελεστές σε προγράμματα, επειδή αυτοί οι τελεστές μπορούν να λειτουργήσουν μόνο σε κλασικούς τύπους δεδομένων, όπως ακέραιους 32 bit και 64 bit. Οι αριθμοί που εμπλέκονται στους αλγόριθμους RSA είναι τυπικά μεγαλύτεροι από 512 bit. Για παράδειγμα, για να πολλαπλασιάσουμε δύο αριθμούς ακέραιους 32-bit a και b, πρέπει απλώς να χρησιμοποιήσουμε a^b στο πρόγραμμά μας. Ωστόσο, εάν οι αριθμοί είναι μεγάλοι, δε μπορούμε να το κάνουμε αυτό. Αντίθετα, πρέπει να χρησιμοποιήσουμε έναν αλγόριθμο (δηλαδή μια συνάρτηση) για να υπολογίσουμε τα γινόμενά τους και θα μας επιτρέψει έτσι να ξεπεράσουμε τα όρια της μηχανής και συγκεκριμένα των κυκλωμάτων της αριθμητικής και λογικής μονάδας της.

Υπάρχουν διάφορες βιβλιοθήκες που μπορούν να εκτελούν αριθμητικές πράξεις σε ακέραιους αριθμούς αυθαίρετου μεγέθους. Σε αυτό το εργαστήριο, θα χρησιμοποιήσουμε τη βιβλιοθήκη μεγάλων αριθμών που παρέχεται από το openssl. Για να χρησιμοποιήσετε αυτήν τη βιβλιοθήκη, θα ορίσουμε κάθε μεγάλο αριθμό ως τύπο BIGNUM και, στη συνέχεια, θα χρησιμοποιήσουμε τα API που παρέχει η βιβλιοθήκη για διάφορες λειτουργίες, όπως πρόσθεση, πολλαπλασιασμό, ύψωση σε δύναμη, αριθμητική υπολογίση.

Στην άσκηση αυτή θα χρησιμοποιηθεί όλο το υλικό του θεωρητικού μέρους του μαθήματος για τον RSA που είναι ανεβασμένο στο eclass.

2.1 BIGNUM APIs

Όλα τα APIs για μεγάλους αριθμούς μπορούν να βρεθούν στο <https://linux.die.net/man/3/bn>. Στη συνέχεια περιγράφονται μερικά από τα APIs που χρειάζονται για την εργαστηριακή αυτή άσκηση.

Ορισμένες συναρτήσεις της βιβλιοθήκης απαιτούν προσωρινές μεταβλητές. Δεδομένου ότι η δυναμική κατανομή μνήμης για τη δημιουργία των BIGNUMs είναι αρκετά δαπανηρήταν χρησιμοποιείται σε συνδυασμό με επαναλαμβανόμενες κλήσεις υπορουτίνας, δημιουργείται μια δομή BNX CTX για να κρατάει προσωρινές μεταβλητές BIGNUM που χρησιμοποιούνται από συναρτήσεις βιβλιοθήκης. Πρέπει να δημιουργήσουμε μια τέτοια δομή και να τη μεταβιβάσουμε (περάσουμε) στις συναρτήσεις που την απαιτούν.

```
BN_CTX *ctx = BN_CTX_new()
```

Αρχικοποίηση της μεταβλητής BIGNUM

```
BIGNUM *a = BN_new()
```

Υπάρχουν πολλοί τρόποι να εκχωρηθεί μια τιμή σε μία BIGNUM μεταβλητή.

```
// Assign a value from a decimal number string
BN_dec2bn(&a, "1234567890112231223");
// Assign a value from a hex number string
BN_hex2bn(&a, "2A3B4C55FF77889AED3F");
// Generate a random number of 128 bits
BN_rand(a, 128, 0, 0);
// Generate a random prime number of 128 bits
BN_generate_prime_ex(a, 128, 1, NULL, NULL, NULL);
```

Εκτύπωση ενός μεγάλου αριθμού.

```
void printBN(char *msg, BIGNUM * a)
{
    // Convert the BIGNUM to number string
    char * number_str = BN_bn2dec(a);
    // Print out the number string
    printf("%s %s\n", msg, number_str);
    // Free the dynamically allocated memory
    OPENSSL_free(number_str);
}
```

Υπολογισμός $res = a - b$:

```
BN_sub(res, a, b);
BN_add(res, a, b);
```

Υπολογισμός $res = a * b$. Απαιτείται μία BN CTX δομή σε αυτό το API.

```
BN_mul(res, a, b, ctx)
```

Υπολογισμός $res = a * b * mod(n)$:

```
BN_mod_mul(res, a, b, n, ctx)
```

Υπολογισμός $res = a^c * mod(n)$:

```
BN_mod_exp(res, a, c, n, ctx)
```

Υπολογισμός αντίστροφου, δηλαδή, για ένα a να βρεθεί b , τέτοιο ώστε $a * b * mod(n) = 1 (a * b \equiv 1 \pmod{n})$. Η τιμή b καλείται ο αντίστροφος του a , στην αριθμητική υπολογίστη που n και υπάρχει μόνο αν $\text{gcd}(a,n)=1$.

```
BN_mod_inverse(b, a, n, ctx);
```

2.2 Παράδειγμα

Στη συνέχεια παρουσιάζεται ένα ολοκληρωμένο παράδειγμα. Σε αυτό, αρχικοποιούμε τρεις BIGNUM μεταβλητές, a , b , και n , και στη συνέχεια υπολογίζουμε το $a * b$ και το $(a * b * mod(n))$. Υπάρχουν πολλοί τρόποι να εκχωρηθεί μια τιμή σε μία BIGNUM μεταβλητή.

```
/* bn_sample.c */
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
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 *a = BN_new();
    BIGNUM *b = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *res = BN_new();

    // Initialize a, b, n
    BN_generate_prime_ex(a, NBITS, 1, NULL, NULL, NULL);
    BN_dec2bn(&b, "273489463796838501848592769467194369268");
    BN_rand(n, NBITS, 0, 0);

    // res = a*b
    BN_mul(res, a, b, ctx);
    printBN("a * b = ", res);

    // res = a^b mod n
    BN_mod_exp(res, a, b, n, ctx);
    printBN("a^b mod n = ", res);
    return 0;
}

```

Μεταγλώττιση: Για τη μεταγλώττιση του προγράμματος bn_sample.c χρησιμοποιείστε την παρακάτω εντολή

```
$ gcc bn_sample.c -lcrypto
```

3 Δραστηριότητες Εργαστηρίου

Για την αποφυγή λαθών θα πρέπει να αντιγράφετε τους αριθμούς που θα συναντήσετε παρακάτω και όχι να τους πληκτρολογείτε. Στην άσκηση που θα παραδώσετε θα πρέπει να συμπεριλάβετε τα βήματα που ακολουθήσατε, τον κώδικα, και τα αποτελέσματα που πήρατε όταν τον τρέξατε.

3.1 Δραστηριότητα 1: Δημιουργώντας το ιδιωτικό κλειδί

Έστω p και q δύο πρώτοι αριθμοί και e ένας αριθμός. Έστω $n = p * q$. Θα χρησιμοποιήσουμε το (e, n) ως το δημόσιο κλειδί. Υπολογίστε το ιδιωτικό κλειδί d . Οι δεκαεξαδικές τιμές των p, q , και e αναφέρονται παρακάτω. Πρέπει να σημειωθεί ότι α και τ p και q που χρησιμοποιούνται σε αυτό τη δραστηριότητα είναι αρκετά μεγάλοι αριθμοί, δεν είναι αρκετά μεγάλοι για να είναι ασφαλείς. Προφανώς τα κάνουμε μικρά για λόγους απλότητας. Στην πράξη, αυτοί οι αριθμοί θα πρέπει να έχουν μήκος τουλάχιστον 512 bits (αυτός που χρησιμοποιείται εδώ είναι μόνο 128 bits).

$$p = F7E75FDC469067FFDC4E847C51F452DF$$

$$q = E85CED54AF57E53E092113E62F436F4F$$

$$e = 0D88C3$$

Απάντηση:

Στην ασύμμετρη κρυπτογράφηση χρησιμοποιούνται δυο κλειδιά κρυπτογράφησης, ένα Ιδιωτικό (Privet Key) που είναι γνωστό μόνο στον ιδιοκτήτη του και ένα Δημόσιο (Public Key) που διατίθεται προς όλους. Η σχέση των κλειδιών είναι τέτοια ώστε όταν κρυπτογραφηθεί κάτι με το ένα να αποκρυπτογραφηθεί αποκλειστικά μόνο με το άλω. Το κάθε κλειδί από αυτά αποτελείται από ένα ζεύγος αριθμόν. Για παράδειγμα το Δημόσιο κλειδί αποτελείται από το ζευγάρι των ακεραίων (e, n) και το Ιδιωτικό από το (d, n) . Το n προκύπτει από το γινωμένο δυο πολύ μεγάλων πρώτων αριθμών p και q . Το e είναι επίσης πρώτος αριθμός και επιλέγετε ώστε να μην έχει κοινούς παράγοντες με το $(p-1)*(q-1)$. Τέλος για το d η επιλογή γίνεται ώστε

$$\begin{aligned} e * d &\equiv 1 * \text{mod}((p - 1) * (q - 1)) \Rightarrow \\ e * d * \text{mod}((p - 1) * (q - 1)) &= 1 \end{aligned}$$

Επομένως για να υπολογίσουμε το ιδιωτικό μας κλειδί θα χρησιμοποιήσουμε μερικά από τα **BIGNUM APIs** για να δηλώσουμε μεταβλητές τύπου **BIGNUM** καθώς και να τους εκχωρήσουμε τιμές. Αρχικά δηλώνουμε της p, q, e, n, d της οποίες θα χρειαστούμε για των υπολογισμών του κλειδιού δηλώνουμε μια εξτρα μεταβλητή οπε για να αποθηκεύσουμε τον αριθμό 1. Αυτό το κάνουμε γιατί η συνάρτηση **BN_sub()** δέχεται ως ορίσματα μόνο **BIGNUM** μεταβλητές. Στην Συνεχεία δηλώνουμε τα q_minus_1 , p_minus_1 και τέλος την n_res . Είμαστε πλέων έτοιμοι

για να αστικοποιήσουμε της τιμές που μας δίνονται με την συνάρτηση BN_hex2bn(). Υπολογίζουμε το n και τα q-1, p-1 καθώς και το γινόμενο τους, οπού το αποτέλεσμα του το αποθηκεύουμε στην n_res. Τέλος με την συνάρτηση BN_mod_inverse() υπολογίζουμε το τελικό d.



```
create_privet_key.c ● Makefile x
56 void printBN(char *msg, BIGNUM * a) {
57     char * number_str = BN_bn2hex(a);
58     printf("%s %s\n", msg, number_str);
59     OPENSSL_free(number_str);
60 }
61 int main(int argc, char const *argv[]){
62     BN_CTX *ctx = BN_CTX_new();
63     BIGNUM *p = BN_new();
64     BIGNUM *q = BN_new();
65     BIGNUM *e = BN_new();
66     BIGNUM *n = BN_new();
67     BIGNUM *q_minus_1 = BN_new();
68     BIGNUM *p_minus_1 = BN_new();
69     BIGNUM *n_res = BN_new();
70     BIGNUM *d = BN_new();
71     BIGNUM *one = BN_new();
72
73     BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
74     BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
75     BN_hex2bn(&e, "0D88C3");
76     BN_hex2bn(&one, "1");
77
78     printBN("p= ",p);
79     printBN("q= ",q);
80     printBN("e= ",e);
81     printBN("one= ",one);
82     BN_mul(n, p, q, ctx);
83     printBN("n= ",n);
84
85     BN_sub(q_minus_1, q, one);
86     BN_sub(p_minus_1, p, one);
87     BN_mul(n_res, p_minus_1, q_minus_1, ctx);
88     BN_mod_inverse(d, e, n_res, ctx);
89     printBN("d= ",d);
90     return 0;
91 }
```

```

lardianos ~/Desktop/asfalia
lardianos ~/Desktop/asfalia 80x24

lardianos > beastbox > ~/Desktop/asfalia > make privet
gcc -o create_privet_key create_privet_key.c -lcrypto
./create_privet_key
p= F7E75FDC469067FFDC4E847C51F452DF
q= E85CED54AF57E53E092113E62F436F4F
e= 0D88C3
one= 01
n= E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
d= 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AE8

```

3.2 Δραστηριότητα 2: Κρυπτογράφηση μηνύματος

Έστω (e , n) το δημόσιο κλειδί. Κρυπτογραφήστε το μήνυμα “A top secret!” (χωρίς τα εισαγωγικά). Πρέπει να μετατρέψουμε αυτή την συμβολοσειρά ASCII σε μια δεκαεξαδική συμβολοσειρά και, στη συνέχεια, να μετατρέψουμε την δεκαεξαδική συμβολοσειρά σε BIGNUM χρησιμοποιώντας το hex-to-bn API BN_hex2bn(). Η ακόλουθη εντολή python μπορεί να χρησιμοποιηθεί για τη μετατροπή μιας απλής συμβολοσειράς ASCII σε μια δεκαεξαδική συμβολοσειρά.

```
$ python -c 'print("A top secret!".encode("hex"))'
```

```
4120746f702073656372657421
```

Τα δημόσια κλειδιά παρατίθενται παρακάτω (σε δεκαεξαδική μορφή). Παρέχουμε επίσης το ιδιωτικό κλειδί d για να επαληθεύσετε το αποτέλεσμα της κρυπτογράφησης.

```

n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
e = 010001 (this hex value equals to decimal 65537)
M = A top secret!
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

```

Απάντηση:

Για την κρυπτογράφηση ενός μηνύματος χρειαζόμαστε το δημόσιο κλειδί του παραλήπτη, το οποίο μας δίνετε (e,n). Το μήνυμα που θα κρυπτογραφήσουμε θα είναι το "A top secret!". Για να γνωρίζουμε ώμος ότι το μήνυμα κρυπτογραφήθηκε σωστά και ότι κρατήθηκε ακέραιο κατά την διαδικασία της κρυπτογράφησης θα πρέπει να μπορέσουμε να το αποκρυπτογραφήσουμε. Εφόσον κρυπτογραφηθεί με το δημόσιο κλειδί μπορεί να αποκρυπτογραφηθεί μόνο με το ιδιωτικό (d,n).

Για την κρυπτογράφηση ισχύει:

$$C = M^e \text{mod}(n)$$

Καθώς και για την αποκρυπτογράφηση:

$$M = C^d \text{mod}(n)$$

Αρχικά μετατρέπουμε το μήνυμα σε δεκαεξαδική μορφή. Έπειτα το εκχωρούμε σε μια μεταβλητή μαζί με τα υπόλοιπα δεδομένα. Στην συνεχεία καλούμε την BN_mod_exp() με ορίσματα το μήνυμα σε δεκαεξαδική μορφή, το δημόσιο κλειδί (e,n) και έναν BN_CTX pointer. Αυτό μας επιστρέφει στην μεταβλητή crypto_message το κρυπτογράφημα. Σε αυτό το σημείο δεν γνωρίζουμε αν το αποτέλεσμα που πήραμε είναι σωστό. για αυτόν τον λόγο θα κάνουμε την ανάστροφη διαδικασία για να δούμε αν αποκρυπτογραφηθεί. Θα χρησιμοποιήσουμε δηλαδή το ιδιωτικό κλειδί (d,n) για να αποκρυπτογραφήσουμε το μήνυμα με τον ίδιο τρόπο, καλώντας την BN_mod_exp() άλλα αυτήν την φορά με ορίσματα οπός, το κρυπτογραφημένο μήνυμα το ιδιωτικό κλειδί (d,n) και τον BN_CTX pointer. όπως παρατηρούμε το αποτέλεσμα είναι το αρχικό μήνυμα.

The terminal window shows the following session:

```
lardianos@beastbox:~/Desktop/asfalia$ make crypto
python -c 'print("A top secret!".encode("hex"))'
4120746f702073656372657421
gcc -o crypto_message crypto_message.c -lcrypto
./crypto_message
n=  DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
e=  010001
d=  74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
crypto message =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75ACDC5DE5FC5FADC
decripted message =  4120746f702073656372657421
python -c 'print("4120746f702073656372657421".decode("hex"))'
A top secret!
```

3.3 Δραστηριότητα 3: Αποκρυπτογράφηση μηνύματος

Το δημόσιο και το ιδιωτικό κλειδί που χρησιμοποιήθηκαν σε αυτή τη δραστηριότητα είναι τα ίδια με αυτά στη δραστηριότητα 2. Αποκρυπτογραφήστε το ακόλουθο κρυπτογράφημα C, και μετατρέψτε το πίσω σε μία ASCII συμβολοσειρά σε αναγνώσιμη μορφή.

C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBDFC7DCB67396567EA1E2493F

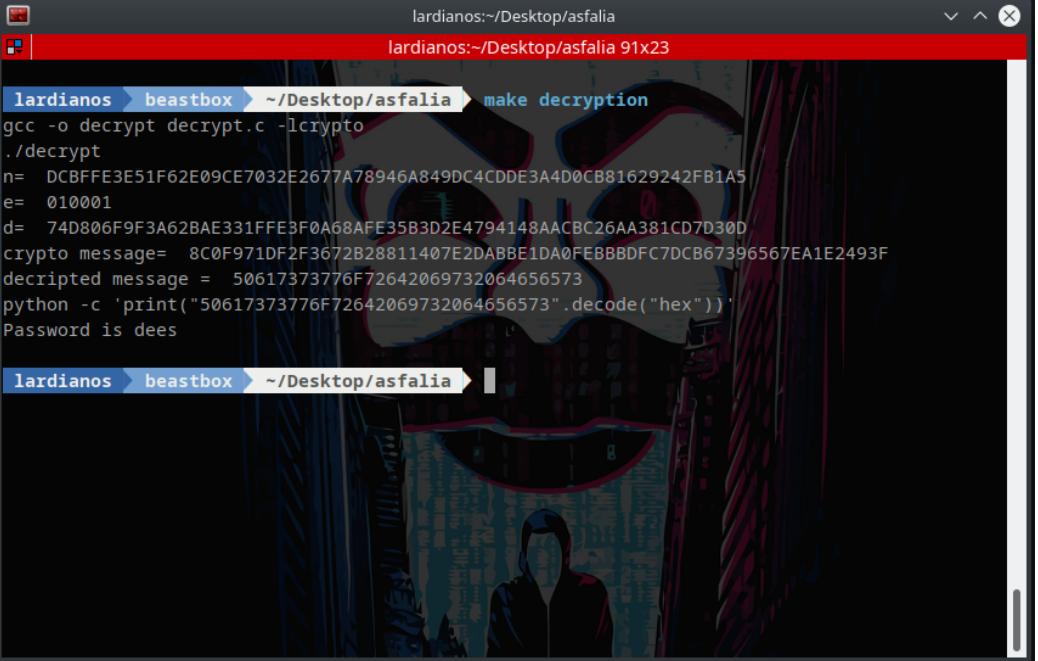
Μπορείτε να χρησιμοποιήσετε την παρακάτω εντολή python για να μετατρέψτε μία δεκαεξαδική συμβολοσειρά πίσω σε μία ASCII συμβολοσειρά σε αναγνώσιμη μορφή.

```
$ python -c 'print("4120746f702073656372657421".decode("hex"))'
```

A top secret!

Απάντηση:

Εφόσον τα κλειδιά είναι ίδια με τα προηγούμενα για την αποκρυπτογράφηση αυτού του μηνύματος θα εφαρμόσουμε την ίδια διαδικασία αποκρυπτογράφησης. μετατρέποντας την δεκαεξαδική συμβολοσειρά σε ascii με την εντολή της python διαβάζουμε το μήνυμα που είχε κρυπτογραφηθεί.



```

169 lardianos@beastbox:~/Desktop/asfalia$ make decryption
170 gcc -o decrypt decrypt.c -lcrypto
171 ./decrypt
172 n= DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
173 e= 010001
174 d= 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
175 crypto message= 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F
176 decrypted message = 50617373776F72642069732064656573
177 python -c 'print("50617373776F72642069732064656573".decode("hex"))
178 Password is dees

```

3.4 Δραστηριότητα 4: Υπογραφή μηνύματος

Το δημόσιο και το ιδιωτικό κλειδί που χρησιμοποιήθηκαν σε αυτή τη δραστηριότητα είναι τα ίδια με αυτά στη δραστηριότητα 2. Δημιουργήστε μία υπογραφή για το ακόλουθο μήνυμα (υπογράψτε απευθείας το μήνυμα αυτό αντί να υπογράψτε την τιμή κατακερματισμού, hash value):

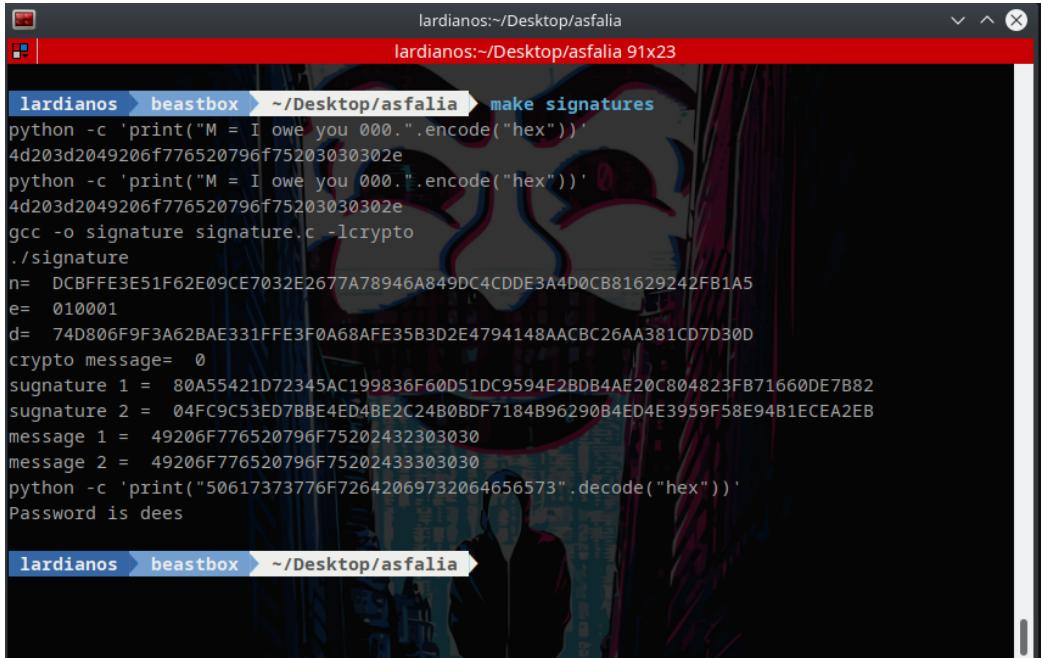
M = I owe you \$2000.

Μετά κάντε μία μικρή αλλαγή στο μήνυμα M, όπως να αλλάξετε το \$2000 σε \$3000, και υπογράψτε το τροποποιημένο μήνυμα. Συγκρίνετε τις υπογραφές και περιγράψτε τι παρατηρείτε.

Απάντηση:

Εφόσον έχουμε κρυπτογραφήσει ένα μήνυμα με το δημόσιο κλειδί κάποιου και του το αποστείλουμε αυτός θα είναι ο μόνος που θα μπορεί να αποκρυπτογράφηση, άλλα σε καμιά περίπτωση δεν θα μπορεί να είναι σίγουρος για το ποιος του το έχει αποστείλει. Με λίγα λογία έχουμε εξασφάλιση την **Ακεραιότητα(integrity)** του μηνύματος και την **Εμπιστευτικότητα(confidentiality)** άλλα όχι την **Αυθεντικότητα(authentication)** και την **Μη άρνηση ταυτότητας (non-repudiation)**. Για να τα εξασφαλίσουμε αυτά χρησιμοποιήσουμε την ψηφιακή υπογραφή. Η ψηφιακή υπογραφή είναι είτε ένα μέρος του μηνύματος είτε ολόκληρο το μήνυμα είτε άπλα κάτι χαρακτηριστικό κρυπτογραφημένο με τα ιδιωτικό κλειδί του αποστολέα και τοποθετημένο συνήθως στο τέλος του κρυπτογραφήματος. εφόσον είναι κρυπτογραφημένο με το ιδιωτικό κλειδιά το αποστολέα μπορεί να αποκρυπτογραφηθεί μόνο από το δημόσιο του κλειδί. Με άπλα λογία μπορεί να αποκρυπτογραφηθεί από οποιονδήποτε έχει λάβει το μήνυμα και εφόσον αποκρυπτογραφηθεί από το δημόσιο κλειδί του αποστολέα σημαίνει ότι ο μοναδικός που θα μπορούσε να το έχει κρυπτογραφήσει θα ήταν ο αποστολέας. Έτσι μπορεί να εξασφαλιστεί και η αυθεντικότητα του μηνύματος άλλα και η μη άρνηση ταυτότητας.

Αρχικά μετατρέπουμε το μήνυμα σε δεκαεξαδική μορφή με το \$2000 και το \$3000. Αν συγκρίνουμε τα αποτελέσματα παρατηρούμε μια διάφορα ενός χαρακτήρα. κρυπτογραφώντας ώμος τα μηνύματα τα κρυπτογραφήματα είναι εντελώς διαφορετικά.



The terminal window shows the following session:

```

lardianos:~/Desktop/asfalia lardianos:~/Desktop/asfalia 91x23
lardianos ~ ~/Desktop/asfalia make signatures
python -c 'print("M = I owe you 000.".encode("hex"))'
4d203d2049206f776520796f752030302e
python -c 'print("M = I owe you 000.".encode("hex"))'
4d203d2049206f776520796f752030302e
gcc -o signature signature.c -lcrypto
./signature
n= DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
e= 010001
d= 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
crypto message= 0
sugnature 1 = 80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82
sugnature 2 = 04FC9C53ED7B8E4E4BE2C4B0BDF7184B96290B4ED4E3959F58E94B1CEA2EB
message 1 = 49206F776520796F75202432303030
message 2 = 49206F776520796F75202433303030
python -c 'print("50617373776F72642069732064656573".decode("hex"))'
Password is dees

```

3.5 Δραστηριότητα 5: Επαλήθευση Υπογραφής

Ο Bob λαμβάνει ένα μήνυμα $M = \text{"Launch a missile."}$ από την Alice, με την υπογραφή της S. Γνωρίζουμε ότι το δημόσιο κλειδί της Alice είναι το (e, n). Επιβεβαιώστε ότι η υπογραφή είναι της Alice ή όχι. Το δημόσιο κλειδί και η υπογραφή (σε δεκαεξαδική μορφή) δίνονται παρακάτω:

$M = \text{Launch a missle.}$

$S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F$

$e = 010001$ (this hex value equals to decimal 65537)

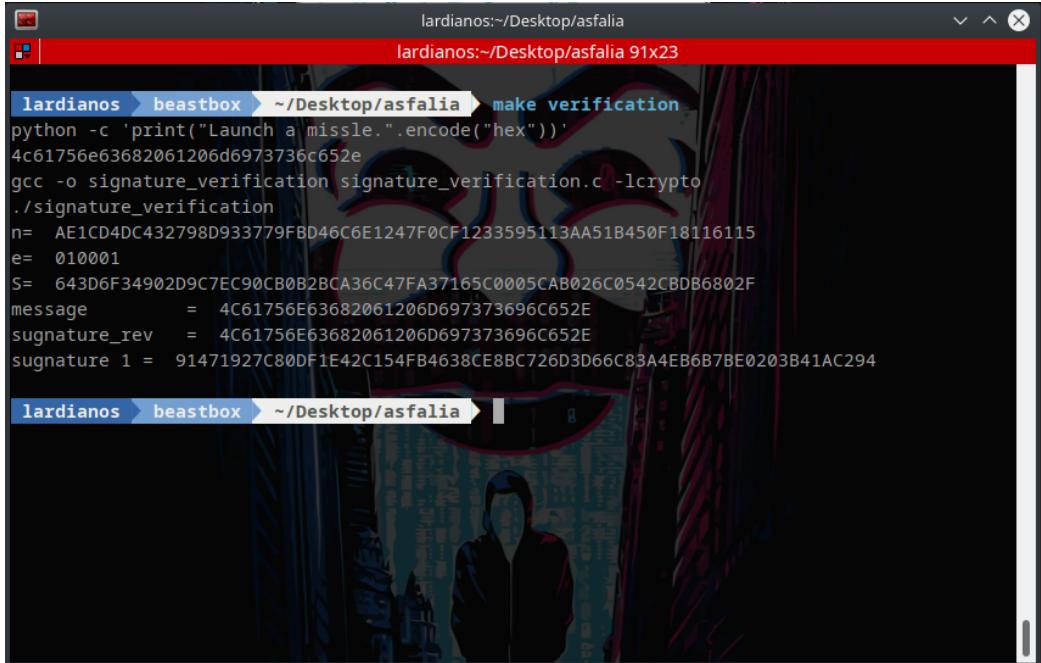
$n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115$

Θεωρείστε ότι η υπογραφή έχει αλλοιωθεί (καταστραφεί), έτσι ώστε το τελευταίο byte της υπογραφής να αλλάζει από 2F σε 3F, δηλαδή, υπάρχει μόνο ένα bit που έχει αλλάξει. Επαναλάβατε τη δραστηριότητα αυτή και περιγράψτε τι θα συμβεί κατά τη διαδικασία επαλήθευσης της υπογραφής.

Απάντηση:

Λαμβάνοντας ο bob το μήνυμα με την ψηφιακή υπογραφή το μόνο που έχει να κάνη για να επιβεβαίωση ότι του το έστειλε η alice είναι να χρησιμοποιήση το δημόσιο κλειδί της για να την αποκρυπτογραφήσει. Αποκρυπτογράφωντας το παρατηρούμε μια δεκαεξαδική συμβολοσειρά χρησιμοποιώντας και πάλι την εντολή της python είτε για να μετατρέψουμε το μήνυμα σε δεκαεξαδικό είτε για να μετατρέψουμε το αποκρυπτογραφημένο δεκαεξαδικό σε ascii. παρατηρούμε ότι είναι ακριβώς τα ίδια, επόμενος σίγουρα αυτό το μήνυμα το έστειλε η alice

Αλλάζοντας μόνο ένα bit από την ψηφιακή υπογραφή παρατηρούμε ότι η συμβολοσειρά που προκύπτει από τη αποκρυπτογραφήσει είναι εντελώς άσχετη με το αρχικό μήνυμα και δεν βγάζει κανένα νόημα αν την μετατρέψουμε σε ascii. αυτό σημαίνει ότι αν η ψηφιακή υπογραφή αλλοιωθεί τότε δεν θα βγάζει κανένα αποτέλεσμα, κάτι που μας εξασφαλίζει την ακεραιότητα ενός μηνύματος



The terminal window shows the command being run and the resulting output:

```
lardianos:~/Desktop/asfalia lardianos:~/Desktop/asfalia 91x23
lardianos > beastbox > ~/Desktop/asfalia > make verification
python -c 'print("Launch a missle.".encode("hex"))
4c61756e63682061206d6973736c652e
gcc -o signature_verification signature_verification.c -lcrypto
./signature_verification
n= AE1CD4DC432798D933779FB046C6E1247F0CF1233595113AA51B450F18116115
e= 010001
S= 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F
message      = 4C61756E63682061206d697373696C652E
signature_rev = 4C61756E63682061206D697373696C652E
signature 1 = 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
```

3.6 Δραστηριότητα 6: Μη αυτόματη επαλήθευση πιστοποιητικού X.509

Σε αυτήν τη δραστηριότητα, θα ελέγξουμε χειροκίνητα ένα πιστοποιητικό X.509 χρησιμοποιώντας το πρόγραμμά μας. Ένα X.509 περιέχει δεδομένα σχετικά με ένα δημόσιο κλειδί και την υπογραφή του εκδότη στα δεδομένα. Θα λάβουμε ένα πραγματικό πιστοποιητικό X.509 από έναν διακομιστή ιστού, θα πάρουμε το δημόσιο κλειδί του εκδότη του και στη συνέχεια θα χρησιμοποιήσουμε αυτό το δημόσιο κλειδί για να επαληθεύσουμε την υπογραφή στο πιστοποιητικό.

Βήμα 1: Κατεβάστε ένα πιστοποιητικό από έναν πραγματικό server. Χρησιμοποιούμε τον διακομιστή www.example.org. Οι φοιτητές θα πρέπει να επιλέξουν διαφορετικό server που έχει διαφορετικό πιστοποιητικό από αυτό που χρησιμοποιείται σε αυτό το έγγραφο (θα πρέπει να σημειωθεί ότι το www.example.com μοιράζεται το ίδιο πιστοποιητικό με το www.example.org). Μπορούμε να κατεβάσουμε πιστοποιητικά χρησιμοποιώντας προγράμματα περιήγησης ή να χρησιμοποιήσουμε την παρακάτω εντολή:

```
$ openssl s_client -connect www.example.org:443 -showcerts
Certificate chain
0 s:/C=US/ST=California/L=Los Angeles/O=Internet Corporation for
Assigned
Names and Numbers/OU=Technology/CN=www.example.org
i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 High
Assurance
Server CA
-----BEGIN CERTIFICATE-----
MIIF8jCCBNqgAwIBAgIQDmTF+8I2reFLFyrrQceMsDANBgkqhkiG9w0BAQsFADBw
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlhnaUNlcnQgSW5jMRkwFwYDVQQLExB3
.....
wDSiIIWIWJiJGbEeIO0TIFwEVWTOnbNI/faPXpk5IRXiapqiII=
-----END CERTIFICATE-----
```

```
1 s:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 High
Assurance Server CA
i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert High
Assurance
EV Root CA
-----BEGIN CERTIFICATE-----
MIIEsTCCA5mgAwIBAgIQBOHnpNxc8vNtwCtCuF0VnzANBgkqhkiG9w0BAQsFADB
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlenQgSW5jMRkwFwYDVQQLExB3
.....
cPUeybQ=
-----END CERTIFICATE-----
```

Το αποτέλεσμα της εντολής περιέχει δύο πιστοποιητικά. Το πεδίο του θέματος (η καταχώριση που αρχίζει με s:) του πιστοποιητικού είναι www.example.org, δηλ. αυτό είναι πιστοποιητικό του www.example.org. Το πεδίο του εκδότη (η καταχώριση που αρχίζει με i:) παρέχει τις πληροφορίες του εκδότη. Το πεδίο του θέματος του δεύτερου πιστοποιητικού είναι το ίδιο με το πεδίο του εκδότη του πρώτου πιστοποιητικού. Βασικά, το δεύτερο πιστοποιητικό ανήκει σε μια ενδιάμεση CA (Certification Authority - Αρχή Πιστοποιήσεων). Σε αυτή τη δραστηριότητα, θα χρησιμοποιήσουμε το πιστοποιητικό της CA για να επαληθεύσουμε ένα πιστοποιητικό διακομιστή.

Εάν λάβετε μόνο ένα πιστοποιητικό πίσω χρησιμοποιώντας την παραπάνω εντολή, αυτό σημαίνει ότι το πιστοποιητικό που λάβατε υπογράφεται από μια root CA. Πιστοποιητικά από root CA μπορούν να ληφθούν από το πρόγραμμα περιήγησης Firefox που είναι εγκατεστημένο στο προκατασκευασμένο VM. Επιλέξτε Edit → Preferences → Privacy → και μετά Security → View Certificates. Αναζητήστε το όνομα του εκδότη και κατεβάστε το πιστοποιητικό του.

Αντιγράψτε και επικολλήστε σε ένα αρχείο καθένα από τα πιστοποιητικά (το κείμενο μεταξύ της γραμμής που περιέχει το “Begin CERTIFICATE” και της γραμμής που περιέχει “End CERTIFICATE”, συμπεριλαμβανομένων των δύο αυτών γραμμών). Ονομάστε το ένα αρχείο c0.pem και το άλλο c1.pem.

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 -modulus. Δεν υπάρχει ειδική εντολή για την εξαγωγή του e, αλλά μπορούμε να εκτυπώσουμε όλα τα πεδία και εύκολα να βρούμε την αξία του e.

For modulus (n):

```
$ openssl x509 -in c1.pem -noout -modulus
Print out all the fields , find the exponent (e):
```

```
$ openssl x509 -in c1.pem -text -noout
```

Βήμα 3: Εξαγάγετε την υπογραφή από το πιστοποιητικό του διακομιστή. 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 μπλοκ υπογραφής σε ένα αρχείο (σημείωση: αν ο αλγόριθμος υπογραφής που χρησιμοποιείται στο πιστοποιητικό δεν βασίζεται στον RSA, μπορείτε να βρείτε ένα άλλο πιστοποιητικό).

```
$ openssl x509 -in c0.pem -text -noout
```

...

```
Signature Algorithm: sha256WithRSAEncryption
```

```
84:a8:9a:11:a7:d8:bd:0b:26:7e:52:24:7b:b2:55:9d:ea:30:
```

```
89:51:08:87:6f:a9:ed:10:ea:5b:3e:0b:c7:2d:47:04:4e:dd:
```

.....

```
5c:04:55:64:ce:9d:b3:65:fd:f6:8f:5e:99:39:21:15:e2:71:
```

```
aa:6a:88:82
```

Πρέπει να αφαιρέσουμε τα διαστήματα και τις άνω κάτω τελείες από τα δεδομένα, έτσι ώστε να έχουμε μια δεκαεξαδική συμβολοσειρά με την οποία μπορούμε να τροφοδοτήσουμε το πρόγραμμά μας. Οι παρακάτω εντολές μπορούν να επιτύχουν αυτόν τον στόχο. Η εντολή tr είναι ένα βοηθητικό εργαλείο Linux για τις λειτουργίες string. Σε αυτή την περίπτωση, η επιλογή -d χρησιμοποιείται για τη διαγραφή των ":" και "κενών" από τα δεδομένα.

```
$ cat signature | tr -d '[:space:]': '
```

```
84a89a11a7d8bd0b267e52247bb2559dea30895108876fa9ed10ea5b3e0bc7
```

.....

```
5c045564ce9db365fdf68f5e99392115e271aa6a8882
```

Βήμα 4: Εξαγάγετε το σώμα του πιστοποιητικού του διακομιστή. Μια Αρχή Πιστοποίησης (CA) δημιουργεί την υπογραφή για ένα πιστοποιητικό διακομιστή, αρχικά υπολογίζοντας το hash του πιστοποιητικού και στη συνέχεια υπογράφει το hash. Για να επαληθεύσουμε την υπογραφή, πρέπει επίσης να δημιουργήσουμε το hash από ένα πιστοποιητικό. Δεδομένου ότι ο καταχερματισμός δημιουργείται πριν από τον υπολογισμό της υπογραφής, πρέπει να αποκλείσουμε το μπλοκ 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.

Τα πιστοποιητικά X.509 κωδικοποιούνται χρησιμοποιώντας το πρότυπο ASN.1 (Abstract Syntax Notation One), οπότε αν μπορέσουμε να αναλύσουμε τη δομή ASN.1, μπορούμε εύκολα να εξαγάγουμε 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

```
$ openssl asn1parse -i -in c0.pem
0:d=0 hl=4 l=1522 cons: SEQUENCE
4:d=1 hl=4 l=1242 cons: SEQUENCE          (1)
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
:0E64C5FBC236ADE14B172AEB41C78CB0
...
1236:d=4 hl=2 l= 12 cons: SEQUENCE
1238:d=5 hl=2 l= 3 prim: OBJECT :X509v3 Basic Constraints
1243:d=5 hl=2 l= 1 prim: BOOLEAN :255
1246:d=5 hl=2 l= 2 prim: OCTET STRING [HEX DUMP]:3000
1250:d=1 hl=2 l= 13 cons: SEQUENCE      (2)
1252:d=2 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption
SEED Labs – RSA Public-Key Encryption and Signature Lab 8
1263:d=2 hl=2 l= 0 prim: NULL
1265:d=1 hl=4 l= 257 prim: BIT STRING
```

Το πεδίο που ξεκινά από το (1) είναι το σώμα του πιστοποιητικού που χρησιμοποιείται για τη δημιουργία του hash. Το πεδίο που ξεκινά από το (2) είναι το μπλοκ της υπογραφής. Οι αποστάσεις τους (offsets) είναι οι αριθμοί στην αρχή των γραμμών. Στην περίπτωσή μας, το σώμα του πιστοποιητικού είναι από το offset 4 έως το 1249, ενώ το μπλοκ υπογραφής είναι από το 1250 έως το τέλος του αρχείου. Για τα πιστοποιητικά X.509, το offset εκκίνησης είναι πάντα το ίδιο (δηλ. 4), αλλά το τέλος εξαρτάται από το μήκος περιεχομένου ενός πιστοποιητικού. Μπορούμε να χρησιμοποιήσουμε την επιλογή -strparse για να πάρουμε το πεδίο από το offset 4, το οποίο θα μας δώσει το σώμα του πιστοποιητικού, εξαιρουμένου του μπλοκ υπογραφής.

```
$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
```

Μόλις λάβουμε το σώμα του πιστοποιητικού, μπορούμε να υπολογίσουμε το hash του χρησιμοποιώντας την ακόλουθη εντολή:

```
$ sha256sum c0_body.bin
```

Βήμα 5: Επαληθεύστε την υπογραφή. Τώρα έχουμε όλες τις πληροφορίες, συμπεριλαμβανομένου του δημόσιου κλειδιού της ΑΠ, της υπογραφής της ΑΠ και του σώματος του πιστοποιητικού του διακομιστή. Μπορούμε να εκτελέσουμε το δικό μας πρόγραμμα για να ελέγξουμε αν η υπογραφή είναι έγκυρη ή όχι. Το Openssl παρέχει μια εντολή για την επαλήθευση του πιστοποιητικού για εμάς, αλλά οι φοιτητές καλούνται να χρησιμοποιήσουν και τα δικά τους προγράμματα για να το κάνουν.

Απάντηση:

Ως πηγή για τα certificates θα χρησιμοποιήσουμε το www.google.com επομένως εκτελούμε την εντολή και μας παρουσιάζονται τα πιστοποιητικά του google.

```

lardianos:~/Desktop/asfalia lardianos:~/Desktop/asfalia 91x23
lardianos ~ beastbox ~ /Desktop/asfalia openssl s_client -connect www.google.com:443 -showcerts
CONNECTED(00000003)
depth=2 OU = GlobalSign Root CA - R2, O = GlobalSign, CN = GlobalSign
verify return:1
depth=1 C = US, O = Google Trust Services, CN = GTS CA 101
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O = Google LLC, CN = www.google.com
verify return:1
-----
Certificate chain
  0 s:C = US, ST = California, L = Mountain View, O = Google LLC, CN = www.google.com
    i:C = US, O = Google Trust Services, CN = GTS CA 101
-----BEGIN CERTIFICATE-----
MIIEvzCCA6egAwIBAgIRAMT1T7RK2JtGCAAAAA4ymcwDQYJKoZIhvcaNAQELBQA
QjELMAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvbsZSBUcnVzdCBTZxJ2aWNlczET
MBEGAlUEAxMRK1RTIENBIDFPMTAeFw0yMDA0MDcwoTQ5MjFaFw0yMDA2MzAwOTQ5
MjFaMGgxCzABgjNQVBYATA1VTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQH
Ew1Nb3VudGFpbjBwahV3MRMwEQYDVQQKEwpHb29nbGUgTExDMRcwFQYDVQQDEw53
d3cu2Z9v2z1LmNbTBZBMGByqGSM49AgECCqGSM49AwEHA0IABKuQr1MtXsKW
wtQZv5BuViOduTXZ2o3JG5MT51CVCRXGNDZPYim3XAMNz+8Y/on043rlfsYTfp
Bt1JB3XY9qjggJTMICtZA0BgNVHQ8Af8EBAMCB4AwEwYDVr0lBAwxCgYIKwYB
BQUHawEwDAYDVROTAQH/BAwADAdBaNVHQ4EFqQU1sDBKJ1/ECxcRepeykT23Jct
-----END CERTIFICATE-----

```

Επιλέγουμε από το BEGIN εως το END και τα αποθηκεύουμε σε δύο αρχεία c0.pem και c1.pem

```

certificates p3.pem ...
p3.pem
-----BEGIN CERTIFICATE-----
MII1DCCGJgAwIBAgIBAAQDQwB0gjwCfzJ527K0gc1IQy5BKTANBgkqhkiG9wBAQ
QjELMAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvbsZSBUcnVzdCBTZxJ2aWNlczET
MBEGAlUEAxMRK1RTIENBIDFPMTAeFw0yMDA0MDcwoTQ5MjFaFw0yMDA2MzAwOTQ5
MjFaMGgxCzABgjNQVBYATA1VTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQH
Ew1Nb3VudGFpbjBwahV3MRMwEQYDVQQKEwpHb29nbGUgTExDMRcwFQYDVQQDEw53
d3cu2Z9v2z1LmNbTBZBMGByqGSM49AgECCqGSM49AwEHA0IABKuQr1MtXsKW
wtQZv5BuViOduTXZ2o3JG5MT51CVCRXGNDZPYim3XAMNz+8Y/on043rlfsYTfp
Bt1JB3XY9qjggJTMICtZA0BgNVHQ8Af8EBAMCB4AwEwYDVr0lBAwxCgYIKwYB
BQUHawEwDAYDVROTAQH/BAwADAdBaNVHQ4EFqQU1sDBKJ1/ECxcRepeykT23Jct
-----END CERTIFICATE-----

```

Εκτελούμε την παρακάτω εντολή και περνούμε το ένα μέρος του δημοσίου κλειδιού (n)

```
lardianos ~/Desktop/asfalia
lardianos ~/Desktop/asfalia 91x23

lardianos beastbox ~/Desktop/asfalia openssl x509 -in p3.pem -noout -modulus
Modulus=DCAE58904DC1C4301590355B6E3C8215F52C5CBDE3DBFF7143FA642580D4EE18A24DF066D00A736E119
8361764AF379DFDFA4184AFC7AF8CFE1A734DCF339790A2968753832BB9A675482D1D56377BDA31321AD7ACAB06
F4AA5D4BB74746DD2A93C3902E798080EF13046A143B59B92BEC207654EFCDAFCFF7AAEDC5C7E55310CE83907A
4D7BE2FD30B6AD2B1DF5FFE5774533B3580DAE8E4498B39F0ED3DAE0D7F46B29AB44A74B58846D924B81C3DA73
8B129748900445751ADD37319792E8CD540D3BE4C13F395E2EB8F35C7E108E8641008D456647B0A165CEA0AA290
94EF397EBE82EAB0F72A7300EFAC7F4FD1477C3A45B2857C2B3F982FDB745589B
```

Για το άλλο μέρος του κλειδιού (e) εκτελούμε τη παρακάτω εντολή και βρίσκουμε το σημείο που γραφή Exponent και το κρατάμε κάπου και αυτό.

```
lardianos ~/Desktop/asfalia
lardianos ~/Desktop/asfalia 91x23

5d:4b:b7:47:46:dd:2a:93:c3:90:2e:79:80:80:ef:
13:04:6a:14:3b:b5:9b:92:be:c2:07:65:4e:fc:da:
fc:ff:7a:ae:dc:5c:7e:55:31:0c:e8:39:07:a4:d7:
be:2f:d3:0b:6a:d2:b1:df:5f:fe:57:74:53:3b:35:
80:dd:ae:8e:44:98:b3:9f:0e:d3:da:e0:d7:f4:6b:
29:ab:44:a7:4b:58:84:6d:92:4b:81:c3:da:73:8b:
12:97:48:90:04:45:75:1a:dd:37:31:97:92:e8:cd:
54:0d:3b:e4:c1:3f:39:5e:2e:b8:f3:5c:7e:10:8e:
86:41:00:8d:45:66:47:b0:a1:65:ce:a0:aa:29:09:
4e:f3:97:eb:e8:2e:ab:0f:72:a7:30:0e:fa:c7:f4:
fd:14:77:c3:a4:5b:28:57:c2:b3:f9:82:fd:b7:45:
58:9b

Exponent: 65537 (0x10001)

X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:TRUE, pathlen:0
    X509v3 Key Usage: critical
        Digital Signature, Certificate Sign, CRL Sign
    Authority Information Access:
        OCSP - URI:http://ocsp.digicert.com

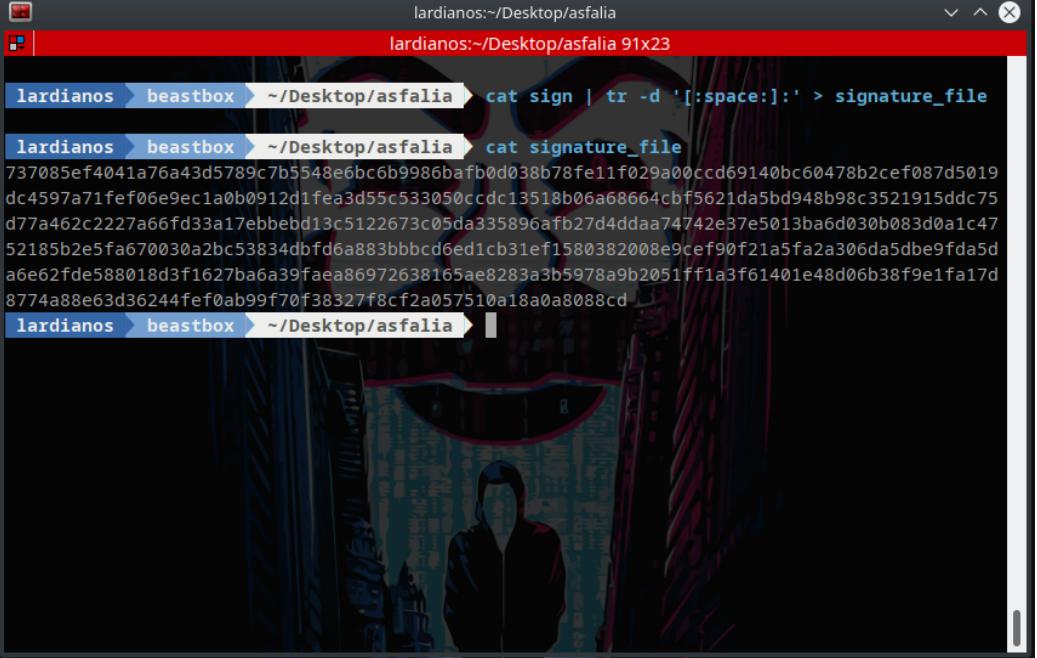
    X509v3 CRL Distribution Points:
        -
```

Εκτελούμε την παρακάτω εντολή αυτήν την φορά για το αρχείο c0.pem και βρίσκουμε το σημείο που γραφή Signature Algorithm και το αντιγράφουμε και αυτό σε ένα αρχείο

```
lardianos:~/Desktop/asfalia
lardianos:~/Desktop/asfalia 91x23
30:45:02:21:00:E4:79:FB:43:84:8E:CA:A1:E4:4F:E9:
03:B0:7A:BB:92:EE:F3:44:3B:8C:EC:FE:14:0D:7D:9F:
B7:63:29:9F:2D:02:20:4D:77:5A:DC:49:01:4A:F4:68:
04:85:61:9F:D7:8D:20:0C:31:FA:C1:D3:F4:71:0A:5B:
D6:56:CB:3D:2C:72:8C
Signature Algorithm: sha256WithRSAEncryption
73:70:85:ef:40:41:a7:6a:43:d5:78:9c:7b:55:48:e6:bc:6b:
99:86:ba:fb:0d:03:8b:78:fe:11:f0:29:a0:0c:cd:69:14:0b:
c6:04:78:b2:ce:f0:87:d5:01:9d:c4:59:7a:71:fe:f0:6e:9e:
c1:a0:b0:91:2d:1f:ea:3d:55:c5:33:05:0c:cd:c1:35:18:b0:
6a:68:66:4c:bf:56:21:da:5b:d9:48:b9:8c:35:21:91:5d:dc:
75:d7:7a:46:2c:22:27:a6:6f:d3:3a:17:eb:be:bd:13:c5:12:
26:73:c0:5d:a3:35:89:6a:fb:27:d4:dd:aa:74:74:2e:37:e5:
01:3b:a6:d0:30:b0:83:d0:a1:c4:75:21:85:b2:e5:fa:67:00:
30:a2:bc:53:83:4d:bf:d6:a8:83:bb:bc:d6:ed:1c:b3:1e:f1:
58:03:82:00:8e:9c:ef:90:f2:1a:5f:a2:a3:06:da:5d:be:9f:
da:5d:a6:e6:2f:de:58:80:18:d3:f1:62:7b:a6:a3:9f:ae:a8:
69:72:63:81:65:ae:82:83:a3:b5:97:8a:9b:20:51:ff:1a:3f:
61:40:1e:48:d0:6b:38:f9:e1:fa:17:d8:77:4a:88:e6:3d:36:
24:4f:ef:0a:b9:9f:70:f3:83:27:f8:cf:2a:05:75:10:a1:8a:
0a:80:88:cd

lardianos > beastbox > ~/Desktop/asfalia >
```

Με την παρακάτω εντολή αναφέρουμε όλα τα κενά και της : που υπάρχουν



```

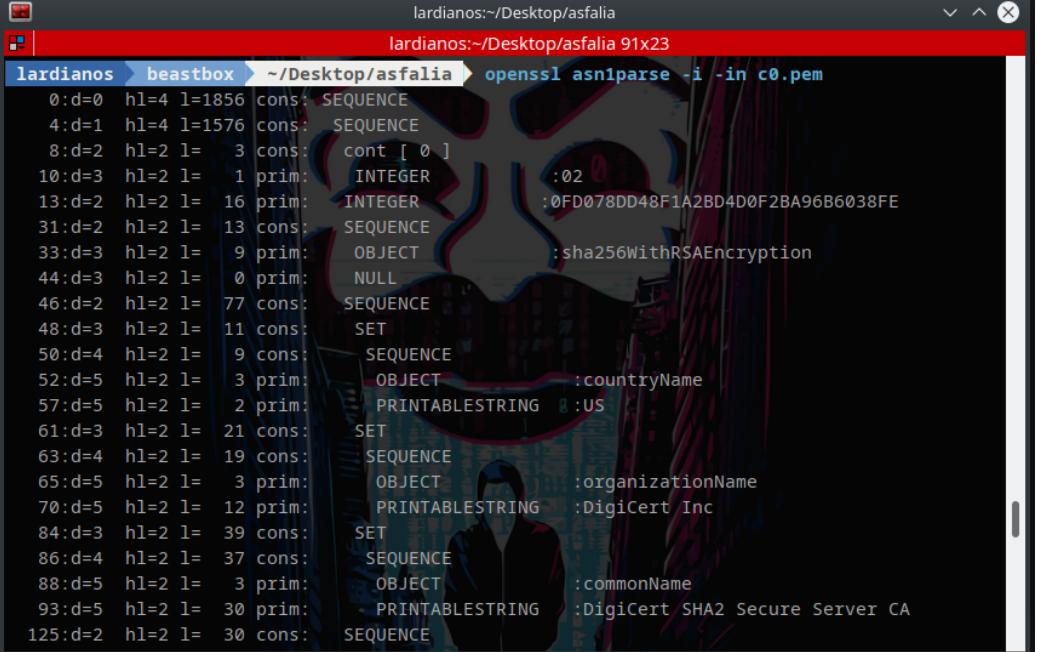
lardianos:~/Desktop/asfalia
lardianos:~/Desktop/asfalia 91x23

lardianos > beastbox > ~/Desktop/asfalia > cat sign | tr -d '[:space:]:' > signature_file

lardianos > beastbox > ~/Desktop/asfalia > cat signature_file
737085ef4041a76aa43d5789c7b5548e6bc6b9986bafb0d038b78fe11f029a00cccd69140bc60478b2cef087d5019
dc4597a71fef06e9ec1a0b0912d1fea3d55c533050ccdc13518b06a68664cbf5621da5bd948b98c3521915ddc75
d77a462c2227a66fd33a17ebbebd13c5122673c05da335896afb27d4ddaa74742e37e5013ba6d030b083d0a1c47
52185b2e5fa670030a2bc53834dbfd6a883bbcd6ed1cb31e1580382008e9cef90f21a5fa2a306da5dbe9fda5d
a6e62fde588018d3f1627ba6a39faea86972638165ae8283a3b5978a9b2051ff1a3f61401e48d06b38f9e1fa17d
8774a88e63d36244fef0ab99f70f38327f8cf2a057510a18a0a8088cd

```

με της παρακάτω εντολές διαχωρίζουμε το σόμα του πιστοποιητικού χωρίς την υπογραφή.

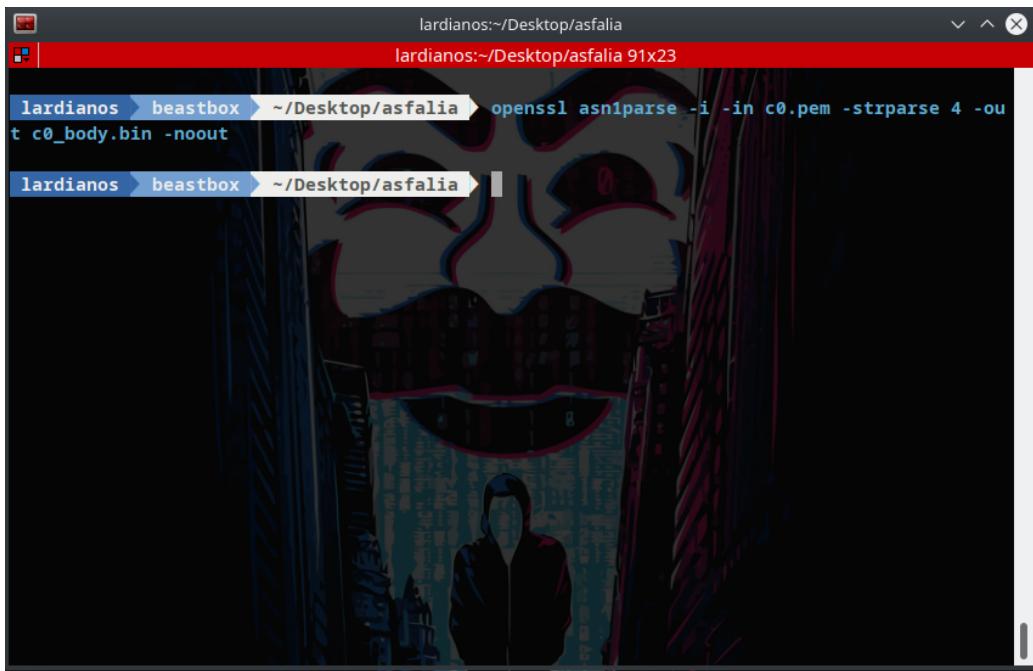


```

lardianos:~/Desktop/asfalia
lardianos:~/Desktop/asfalia 91x23

lardianos > beastbox > ~/Desktop/asfalia > openssl asn1parse -i -in c0.pem
0:d=0 hl=4 l=1856 cons: SEQUENCE
4:d=1 hl=4 l=1576 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 :0FD078DD48F1A2BD4D0F2BA96B6038FE
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= 77 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= 39 cons: SET
86:d=4 hl=2 l= 37 cons: SEQUENCE
88:d=5 hl=2 l= 3 prim: OBJECT :commonName
93:d=5 hl=2 l= 30 prim: PRINTABLESTRING :DigiCert SHA2 Secure Server CA
125:d=2 hl=2 l= 30 cons: SEQUENCE

```

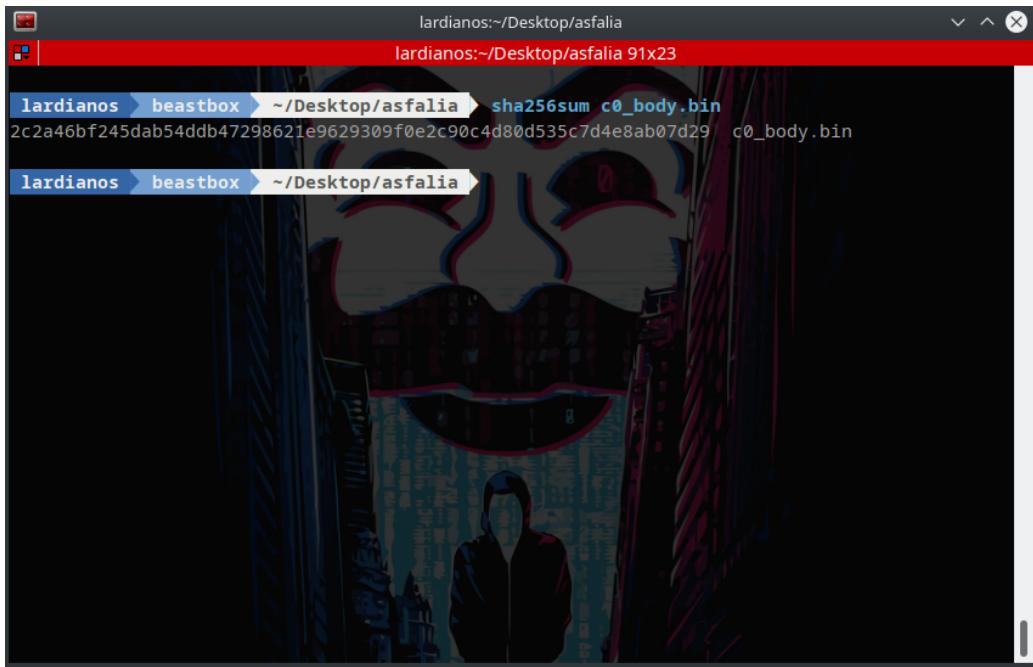


```
lardianos:~/Desktop/asfalia
lardianos:~/Desktop/asfalia 91x23

lardianos beastbox ~/Desktop/asfalia > openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout

lardianos beastbox ~/Desktop/asfalia >
```

Από το σόμα του πιστοποιητικού υπολογίζουμε το hash.



```
lardianos:~/Desktop/asfalia
lardianos:~/Desktop/asfalia 91x23

lardianos beastbox ~/Desktop/asfalia > sha256sum c0_body.bin
2c2a46bf245dab54ddb47298621e9629309f0e2c90c4d80d535c7d4e8ab07d29 c0_body.bin

lardianos beastbox ~/Desktop/asfalia >
```

και πλέων έχουμε όλες της πληροφορίες που χρειαζόμαστε για να κάνουμε την επαλήθευση.

3.6 Δραστηριότητα 6: Μη αυτόματη επαλήθευση πιστοποιητικού X.509

28

```

users_file
1 | google
2 | -----
3 | DCAE58904DC1C4301590355B6E3C8215F52C5CBDE3DBFF7143FA642580D4EE18A24DF06000A73
4 | 6E1198361764AF3790FDFA148AFC7AF8CFE1A734DCF339790A29607532B89A67548201D5637
5 | 78DA31321AD7ACAB0FA5A04B8747460D2A93C3902E79880BF13040A143B859B928EC207654E
6 | FCDACFC7FAAEADC5C7E5310C83997A407BE2F03080AD281D5FF57745338358000AE449883
7 | 9F8ED3D4E807F46822A8444748588460924881CD738B129748890445751A0D037319792EBC054
8 | 038E4C13F395E8EB8F5C7E1888641000045664780A165CEA0AA29894EF337E8E82EAB80F72A7
9 | 388E7AC77AF21477C3A4582657C283F9827087458098
10 | 2c2a46bf245dab54ddb47298621e9629309f0e2c98c4d80d535c7d4e8ab07d29
11 |
12 | Exponent: 65537 (0x10001)
13 |
14 | hash
15 | -----
16 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
17 | 737885ef4041a76a43d5789c7b5548e6bc6b9986baf0d038b78fe11f029a00c
18 | 130d5b705d910c559774ff7089e9c100b912df1fa3d5c533950cc1351bb06ad05641cf758
19 | 21d5b9d94809c35219154dc75d774462c227a6fd53a17e5bebd13c5124079c95da335896af5b
20 | 27d4ddaa74742e37e5013ba6d0300b8300a1c4752185b2e5fa670038a2bc53834dbfd6a83bbcc
21 | d6ed1c31ef1580382008e9cef90f21a5fa2a306da5dbe9fda5da6e62fde588018d3f1627ba6a3
22 | 9faee86f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
23 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
24 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
25 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
26 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
27 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
28 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
29 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
30 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
31 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
32 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
33 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
34 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
35 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
36 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
37 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
38 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
39 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
40 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
41 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
42 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
43 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
44 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
45 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
46 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
47 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
48 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
49 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
50 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
51 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
52 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
53 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
54 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
55 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
56 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
57 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
58 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
59 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
60 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
61 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
62 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
63 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
64 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
65 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
66 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
67 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
68 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
69 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
70 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
71 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
72 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
73 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
74 | 244fe0ab99f72638105ae28ba3059789b2051fffa3f61401e48d0eb38f5e1fa17d8774a88e63d36
75 |

```

Της περνάμε στο πρόγραμμα μας

```

certificates2.c
1 | /* Use BN_bn2dec(a) for decimal string */
2 | char * number_str = BN_bn2hex(a);
3 | printf("%s %s\n", msg, number_str);
4 | OPENSSL_free(number_str);
5 |
6 |
7 | int main(int argc, char const *argv[]){
8 |     BN_CTX *ctxx = BN_CTX_new();
9 |     BIGNUM *e = BN_new();
10 |     BIGNUM *n = BN_new();
11 |     BIGNUM *signature = BN_new();
12 |     BIGNUM *hash = BN_new();
13 |     BIGNUM *S_corrupted = BN_new();
14 |     BIGNUM *signature_rev = BN_new();
15 |
16 |     /* Dimosios Klidi (e,n)
17 |      Idiotiko klidi (d,n)
18 |     */
19 |     BN_hex2bn(&n, "DCAE58904DC1C4301590355B6E3C8215F52C5CBDE3DBFF7143FA642580D4EE18A24DF066");
20 |     printf("\n");
21 |     BN_hex2bn(&e, "010001");
22 |     printf("\n");
23 |     BN_hex2bn(&signature, "737085ef4041a76a43d5789c7b5548e6bc6b9986baf0d038b78fe11f029a00c");
24 |     printf("\n");
25 |     BN_hex2bn(&hash, "2c2a46bf245dab54ddb47298621e9629309f0e2c90c4d80d535c7d4e8ab07d29");
26 |     printf("\n");
27 |     printBN("Public Key = ",n);
28 |     printBN("Exponent= ",e);
29 |     printBN("Signature = ",signature);
30 |     printBN("Hash = ",hash);
31 |
32 |     BN_mod_exp(signature_rev, signature, e, n, ctxx);
33 |     BN_mask_bits(signature_rev, 256);
34 |     printBN("Signature auhenticated = ",signature_rev);
35 |     printf("\n");
36 |
37 |     return 0;
38 }

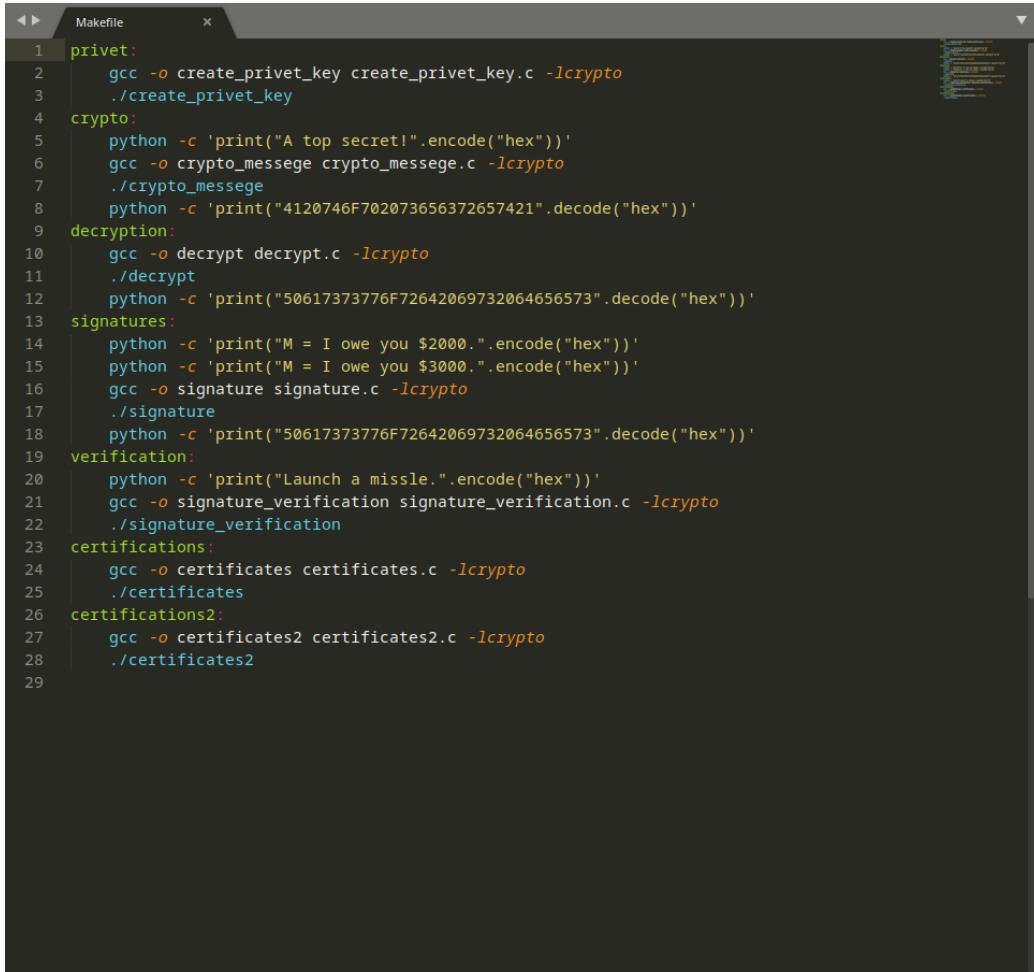
```

Το εκτελούμε και παρατηρούμε ότι το hash που υπολογίσαμε είναι ίδιο με αυτό που προέκυψε από την αποκρυπτογράφηση επομένως η επαλήθευση ήταν επιτυχής!

```
lardianos:~/Desktop/asfalia 91x25
lardianos  beastbox  ~/Desktop/asfalia  make certifications2
gcc -o certificates2 certificates2.c -lcrypto
./certificates2

Public Key = DCAE58904DC1C4301590355B6E3C8215F52C5CBDE3DBFF7143FA642580D4EE18A24DF066D00A7
36E1198361764AF379DFDFA418AFC7AF8CFE1A734DCF339790A2968753832BB9A675482D1D56377BDA31321AD7
ACAB06F4AA5D4BB74746DD2A93C3902E798080EF13046A143BB59B92BEC207654EFCDAFCFF7AAEDC5C7E55310CE
83907A4D7BE2FD30B6AD2B1DF5FFE5774533B3580DDAE8E4498B39F0ED3DAE0D7F46B29AB44A74B58846D924B81
C3DA738B129748900445751ADD37319792E8CD540D3BE4C13F395E2EB8F35C7E108E8641008D456647B0A165CEA
0AA29094EF397E82EAB0F72A7300EFAC7F4FD1477C3A45B2857C2B3F982FDB745589B
Exponent= 010001
Signature      = 737085EF4041A76A43D5789C7B5548E6BC6B9986BAFB0D038B78FE11F029A00CCD69140B
C60478B2CEF087D5019DC4597A71FEF06E9EC1A0B0912D1FEA3D55C533050CCDC13518B06A68664CBF5621DA5BD
948B98C3521915DDC75D77A462C2227A66FD33A17EBBEBD13C5122673C05DA335896AFB27D4DDA74742E37E501
3BA6D030B083D0A1C4752185B2E5FA670030A2BC53834DBFD6A883BBBCD6ED1CB31EF1580382008E9CEF90F21A5
FA2A306DA5DBE9FDA5DA6E62FDE588018D3F1627BA6A39FAEA86972638165AE8283A3B5978A9B2051FF1A3F6140
1E48D06B38F9E1FA17D8774A88E63D36244FEF0AB99F70F38327F8CF2A057510A18A0A8088CD
Hash           = 2C2A46BF245DAB54DDB47298621E9629309F0E2C90C4D80D535C7D4E8AB07D29
Signature auhenticated = 2C2A46BF245DAB54DDB47298621E9629309F0E2C90C4D80D535C7D4E8AB07D29
```

Τέλος για την ευκολία μας μπορούμε να δημιουργήσουμε ένα Makefile σαν το παρακάτω που μας γλιτώνει χρόνο στα compile και execute των προγραμμάτων μας για της δόκιμες που κάναμε.



```
Makefile

1 privet:
2     gcc -o create_privet_key create_privet_key.c -lcrypto
3     ./create_privet_key
4 crypto:
5     python -c 'print("A top secret!".encode("hex"))'
6     gcc -o crypto_messege crypto_messege.c -lcrypto
7     ./crypto_messege
8     python -c 'print("4120746F702073656372657421".decode("hex"))'
9 decryption:
10    gcc -o decrypt decrypt.c -lcrypto
11    ./decrypt
12    python -c 'print("50617373776F72642069732064656573".decode("hex"))'
13 signatures:
14    python -c 'print("M = I owe you $2000.".encode("hex"))'
15    python -c 'print("M = I owe you $3000.".encode("hex"))'
16    gcc -o signature signature.c -lcrypto
17    ./signature
18    python -c 'print("50617373776F72642069732064656573".decode("hex"))'
19 verification:
20    python -c 'print("Launch a missle.".encode("hex"))'
21    gcc -o signature_verification signature_verification.c -lcrypto
22    ./signature_verification
23 certifications:
24    gcc -o certificates certificates.c -lcrypto
25    ./certificates
26 certifications2:
27    gcc -o certificates2 certificates2.c -lcrypto
28    ./certificates2
29
```