RSA και Ψηφιακές Υπογραφές

Εισαγωγή

Ο κρυπτογραφικός αλγόριθμος **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 υπολοίπου που καθιστά αδύνατη την αποκρυπτογράφηση με παραγοντοποίηση. Θεωρείται μέχρι σήμερα ένας ασφαλής κρυπτογραφικός αλγόριθμος.

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 ακέραιους αριθμούς 32-bit **`a`** και **`b`**, πρέπει απλώς να χρησιμοποιήσουμε **`a*b`** στο πρόγραμμά μας. Ωστόσο, εάν οι αριθμοί είναι μεγάλοι, αυτό δεν μπορεί να γίνει. Αντίθετα, πρέπει να χρησιμοποιήσουμε έναν αλγόριθμο (δηλαδή μια συνάρτηση) για να υπολογίσουμε τα γινόμενά τους, επιτρέποντας μας 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 **`bn`** (multiprecision integer arithmetics) που παρέχεται από το **`openssl`**[1]. Η βιβλιοθήκη αυτή παρέχει τον τύπο **`BIGNUM`**[2], ο οποίος μπορεί να αναπαραστήσει κάθε μεγάλο αριθμό. Επίσης, παρέχει ένα API που υλοποιεί διάφορες λειτουργίες, όπως πρόσθεση, πολλαπλασιασμό, ύψωση σε δύναμή, αριθμητική υπολοίπου κ.λπ.

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

[1] https://www.openssl.org
[2] https://www.openssl.org/docs/man1.0.2/man3/bn.html

BIGNUM APIs

Όλα τα APIs για μεγάλους αριθμούς που παρέχει η βιβλιοθήκη bn υπάρχουν στο https://www.openssl.org/docs/man1.0.2/man3/bn.html. Στη συνέχεια της άσκησης, περιγράφονται ορισμένες από τις λειτουργίες που χρειάζονται για την παρούσα εργαστηριακή άσκηση.

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

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

- **Αρχικοποίηση** μιας μεταβλητής BIGNUM:

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

- **Εκχώρηση** τιμής σε μια μεταβλητή BIGNUM: Υπάρχουν πολλοί τρόποι να εκχωρηθεί μια τιμή σε μία BIGNUM μεταβλητή.

```
// Assign a value from a decimal number string
BN_dec2bn(&a, "12345678901112231223");
// 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 **αφαίρεσης** και **πρόσθεσης**:

```
//Calculate res = a - b
BN_sub(res, a, b);
//Calculate res = a + b
BN_add(res, a, b);
```

- Υπολογισμός **πολλαπλασιασμού** (απαιτείται μία δομή `BN_CTX` σε αυτό το API):

```
//Calculate res = a * b
BN_mul(res, a, b, ctx)
```

- Υπολογισμός **`res = a * b mod n`** (απαιτείται μία δομή `BN_CTX` σε αυτό το API):

```
//Calculate res = a * b mod n
BN_mod_mul(res, a, b, n, ctx)
```

- Υπολογισμός **$res = a^c \bmod n$** (απαιτείται μία δομή `BN_CTX` σε αυτό το API):

```
//Calculate res = a^c mod n
BN_mod_exp(res, a, c, n, ctx)
```

- Υπολογισμός **αντίστροφου σε αριθμητική υπολοίπου**, δηλαδή, για ένα **a** να βρεθεί **b**, τέτοιο ώστε να ισχύει:

  `a * b mod n = 1,` δηλαδή **$(a * b \equiv 1 \bmod n)$**.

  Η τιμή **b** καλείται ο αντίστροφος του **a**, στην αριθμητική υπολοίπου **n** και υπάρχει μόνο αν **`gcd(a,b)=1`**.

```
//Calculate mod inverse
BN_mod_inverse(b, a, n, ctx);
```

Παράδειγμα

Στη συνέχεια παρουσιάζεται ένα ολοκληρωμένο παράδειγμα χρήσης της βιβλιοθήκης **bn**. Σε αυτό, αρχικοποιούμε τρεις BIGNUM μεταβλητές, a, b, και n, και στη συνέχεια υπολογίζουμε και τυπώνουμε το **a*b** και το **(a^b mod n)**.

```
/* 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);

  //calculate and print res = a*b
  BN_mul(res, a, b, ctx);
```

```
  printBN("a * b = ", res);

  // calculate and print res = a^b mod n
  BN_mod_exp(res, a, b, n, ctx);
  printBN("a^c mod n = ", res);
  return 0;
}
```

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

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

Δραστηριότητες Εργαστηρίου

Για την εκτέλεση των παρακάτω δραστηριοτήτων θα πρέπει να χρησιμοποιήσετε την εικονική μηχανή **SEED Ubuntu 16.04**. Σε αυτήν θα βρείτε όλες οι απαραίτητες εφαρμογές (*browser, terminal*) καθώς και εργαλεία, όπως *openssl*, *python*.

Δραστηριότητα 1: Δημιουργία ιδιωτικού κλειδιού

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.

```
p = 953AAB9B3F23ED593FBDC690CA10E703
q = C34EFC7C4C2369164E953553CDF94945
e = 0D88C3
```

Σημείωση: Πρέπει να σημειωθεί ότι αν και τα **p** και **q** που χρησιμοποιούνται σε αυτή τη δραστηριότητα είναι μεγάλοι αριθμοί, δεν είναι αρκετά μεγάλοι για να είναι ασφαλείς. Για τις ανάγκες της άσκησης και για λόγους απλότητας χρησιμοποιούμε αριθμούς 128 bits. Στην πράξη, αυτοί οι αριθμοί θα πρέπει να έχουν μήκος τουλάχιστον 512 bits.

Δραστηριότητα 2: Κρυπτογράφηση μηνύματος

Θεωρήστε το μήνυμα **M** που αποτελείται από το ονοματεπώνυμό σας (με λατινικούς χαρακτήρες), με την παρακάτω μορφή:

```
Onoma Eponymo
```

Θα πρέπει να γράψετε τον κώδικα που να κρυπτογραφεί το μήνυμα αυτό. Το δημόσιο κλειδί **(e, n)** που θα χρησιμοποιήσετε θα είναι το ίδιο με αυτό που χρησιμοποιήσατε στη Δραστηριότητα 1. Περαιτέρω, με το ιδιωτικό κλειδί **d** που βρήκατε στη Δραστηριότητα 1, θα πρέπει να επαληθεύσετε το αποτέλεσμα της κρυπτογράφησης.

Σημείωση: Για να προχωρήσετε στην κρυπτογράφηση θα πρέπει πρώτα να μετατρέψετε το μήνυμά σας από συμβολοσειρά ASCII σε δεκαεξαδική (hex) συμβολοσειρά και στη συνέχεια, να μετατρέψετε την δεκαεξαδική συμβολοσειρά σε BIGNUM χρησιμοποιώντας τη συνάρτηση 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
```

Αντίστοιχα, για να μετατρέψτε μία **δεκαεξαδική** συμβολοσειρά σε μία **ASCII** συμβολοσειρά (αναγνώσιμη μορφή) μπορείτε να χρησιμοποιήσετε την παρακάτω εντολή python.

```
$ python –c ’print("4120746f702073656372657421".decode("hex"))’
 A top secret!
```

Δραστηριότητα 3: Αποκρυπτογράφηση μηνύματος

Το δημόσιο και το ιδιωτικό κλειδί που χρησιμοποιούνται σε αυτή τη δραστηριότητα είναι τα εξής:

```
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
e = 010001 (this hex value equals to decimal 65537)
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
```

Αποκρυπτογραφήστε το ακόλουθο κρυπτογράφημά C, και μετατρέψτε το πίσω σε μία ASCII συμβολοσειρά σε αναγνώσιμή μορφή.

```
C= B3AF0A70793BB53492B5311AED5EA843D94661924C97A446E9DD75846DF860DF
```

Δραστηριότητα 4: Υπογραφή μηνύματος

Το δημόσιο και το ιδιωτικό κλειδί που χρησιμοποιούνται σε αυτή τη δραστηριότητα είναι τα ίδια με αυτά στη δραστηριότητα 3. Θεωρήστε ένα σύντομο μήνυμα **M** της επιλογή σας (με λατινικούς χαρακτήρες). Δημιουργήστε μία **ψηφιακή υπογραφή** του μηνύματός σας. Μετά κάντε μία μικρή αλλαγή στο μήνυμα **M,** όπως να αλλάξετε ένα γράμμα ή έναν αριθμό, και υπογράψτε ξανά το τροποποιημένο μήνυμα. Τέλος, συγκρίνετε τις δυο υπογραφές και περιγράψτε τι παρατηρείτε.

Σημείωση: για τις ανάγκες της άσκησης, υπογράψτε **απευθείας το μήνυμα** αυτό αντί να υπογράψτε την τιμή κατακερματισμού (hash value):

Δραστηριότητα 5: Επαλήθευση Υπογραφής

Να κάνετε επαλήθευση υπογραφής στις παρακάτω δύο περιπτώσεις:

Περίπτωση Α

Η Alice λαμβάνει ένα μήνυμα M = "Launch a missile." από τον Bob, μαζί με την υπογραφή του S. Γνωρίζουμε ότι το δημόσιο κλειδί του Bob είναι το **(e, n)**. Επιβεβαιώστε ότι η υπογραφή είναι του Bob ή όχι. Το δημόσιο κλειδί και η υπογραφή (σε δεκαεξαδική μορφή) δίνονται παρακάτω:

```
M = Launch a missile.
S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F
e = 010001 (this hex value equals to decimal 65537)
n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115
```

Στη συνέχεια, 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.

Περίπτωση Β

Ο Bob λαμβάνει το μήνυμα M="Please transfer me $2000.Alice." από την Alice, μαζί 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 = Please transfer me $2000.Alice.
S = DB3F7CDB93483FC1E70E4EACA650E3C6505A3E5F49EA6EDF3E95E9A7C6C7A320
e = 010001 (this hex value equals to decimal 65537)
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
```

(Υπόδειξη: μπορείτε, εάν θέλετε, να χρησιμοποιήσετε και τη συνάρτηση **BN_cmp** για τη σύγκριση δύο αριθμών – βλ. σχετικά και το σύνδεσμο:
https://www.openssl.org/docs/man1.0.2/man3/BN_cmp.html)

Δραστηριότητα 6: Μη αυτόματη επαλήθευση πιστοποιητικού X.509

Σε αυτήν τη δραστηριότητα, θα πρέπει να ελέγξετε χειροκίνητα ένα πιστοποιητικό `X.509` χρησιμοποιώντας το πρόγραμμα που αναπτύξατε σε προηγούμενη δραστηριότητα. Ένα πιστοποιητικό X.509 περιέχει δεδομένα σχετικά με ένα δημόσιο κλειδί και την υπογραφή του εκδότη στα δεδομένα. Θα πρέπει να λάβετε ένα πραγματικό πιστοποιητικό X.509 από **έναν διακομιστή web** της επιλογής σας, να εξάγετε το δημόσιο κλειδί του εκδότη του και στη συνέχεια να χρησιμοποιήσετε αυτό το δημόσιο κλειδί για να επαληθεύσετε την υπογραφή στο πιστοποιητικό.

Βήμα 1: Κατεβάστε ένα πιστοποιητικό από έναν πραγματικό web server **της επιλογής σας**. Μπορείτε να κατεβάσετε πιστοποιητικά χρησιμοποιώντας τον browser ή με `openssl` από το `terminal`, χρησιμοποιώντας την παρακάτω εντολή:

```
$ openssl s_client -connect www.example.org:443 -showcerts
```

Το αποτέλεσμα της εντολής στο συγκεκριμένο παράδειγμα περιέχει δυο πιστοποιητικά. Το πεδίο του **θέματος** (η καταχώριση που αρχίζει με την ένδειξη **`s:`**) του πιστοποιητικού είναι www.example.org, δηλ. αυτό είναι πιστοποιητικό του www.example.org. Το πεδίο του **εκδότη** (η καταχώριση που αρχίζει με την ένδειξη **`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).

```
...
Certificate chain
0  s:/C=US/ST=California/L=Los Angeles/O=Internet Corporation for Assigned Names
       and Numbers/CN=www.example.org
   i:/C=US/O=DigiCert Inc/CN=DigiCert TLS RSA SHA256 2020 CA1
-----BEGIN CERTIFICATE-----
MIIG1TCCBb2gAwIBAgIQD74IsIVNBXOKsMzhya/uyTANBgkqhkiG9w0BAQsFADBP
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMSkwJwYDVQQDEyBE
...
/sgUKGiQxrjIlH/hD4n6p9YJN6FitwAntb7xsV5FKAazVBXmw8isggHOhuIr4Xrk
vUzLnF7QYsJhvYtaYrZ2MLxGD+NFI8BkXw==
-----END CERTIFICATE-----

1  s:/C=US/O=DigiCert Inc/CN=DigiCert TLS RSA SHA256 2020 CA1
   i:/C=US/O=DigiCert Inc/ OU=www.digicert.com/CN=DigiCert Global Root CA
-----BEGIN CERTIFICATE-----
```

```
IIE6jCCA9KgAwIBAgIQCjUI1VwpKwF9+K1lwA/35DANBgkqhkiG9w0BAQsFADBh
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
...
as6xuwAwapu3r9rxxZf+ingkquqTgLozZXq8oXfpf2kUCwA/d5KxTVtzhwoT0JzI
8ks5T1KESaZMkE4f97Q=
-----END CERTIFICATE-----

2  s:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert Global Root CA
   i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert Global Root CA
-----BEGIN CERTIFICATE-----
MIIDrzCCApegAwIBAgIQCDvgVpBCRrGhdWrJWZHHSjANBgkqhkiG9w0BAQUFADBh
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
...
YSEY1QSteDwsOoBrp+uvFRTp2InBuThs4pFsiv9kuXclVzDAGySj4dzp30d8tbQk
CAUw7C29C79Fv1C5qfPrmAESrciIxpg0X40KPMbp1ZWVbd4=
-----END CERTIFICATE-----
```

Εάν χρησιμοποιώντας την παραπάνω εντολή λάβετε **μόνο ένα πιστοποιητικό**, αυτό σημαίνει ότι το πιστοποιητικό που λάβατε υπογράφεται απευθείας από μια **root CA**. Το πιστοποιητικό ενός **root CA** μπορεί να ληφθεί από το Firefox browser που είναι εγκατεστημένος στην εικονική μηχανή. 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 την επιλογή `–modulus`.

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

Δεν υπάρχει ειδική εντολή για την εξαγωγή του **`e`**, αλλά μπορούμε να εκτυπώσουμε όλα τα πεδία και εύκολα να βρούμε την τιμή του e.

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

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).

```
$ 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 του `Linux` για τη διαχείριση `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 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.

Τα πιστοποιητικά **`X.509`** κωδικοποιούνται σύμφωνα με το πρότυπο **ASN.1** (Abstract Syntax Notation.One), οπότε αν μπορέσουμε 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`.

```
$ openssl asn1parse -i -in c0.pem
    0:d=0 hl=4 l=1522 cons: SEQUENCE
    4:d=1 hl=4 l=1242 cons: SEQUENCE          ❶
```

```
    8:d=2 hl=2l= 3cons: 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            ❷
1252:d=2  hl=2 l=   9 prim: OBJECT   :sha256WithRSAEncryption
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 ❶ είναι το σώμα του πιστοποιητικού που χρησιμοποιείται για τη δημιουργία του **hash**. Το πεδίο που ξεκινά από το ❷ είναι το block 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 block 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 εξάγουμε το πεδίο από το *offset 4*, το οποίο θα μας δώσει το σώμα του πιστοποιητικού, εξαιρουμένου του block 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
```

(Προσοχή: το ανωτέρω ισχύει εφόσον η hash function που έχει χρησιμοποιηθεί για την υπογραφή είναι η SHA256. Διαφορετικά, η ως άνω εντολή χρειάζεται προσαρμογή).

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.

Υποβολή

Θα πρέπει να υποβάλετε μια λεπτομερή **εργαστηριακή αναφορά** (σε PDF) που θα καλύπτει όλες τις ενέργειες που κάνατε σε κάθε δραστηριότητα, τα αποτελέσματα και τις παρατηρήσεις σας. Στην περιγραφή σας χρησιμοποιείστε στιγμιότυπα οθόνης και αποσπάσματα κώδικα, σύμφωνα με τις οδηγίες κάθε δραστηριότητας. Επίσης, θα πρέπει να υποβάλετε **σε ξεχωριστά αρχεία** τον (κατάλληλα σχολιασμένο) πηγαίο κώδικα που αναπτύξατε.