Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Υπολογιστών

Εργαστήριο Ασφάλειας στην Τεχνολογία της Πληροφορίας – Εργασία 1

Χρήστος Μαργιώλης – 19390133

Απρίλιος 2022

Περιεχόμενα

1 Δομή αρχείων

1.1 Αρχεία C

Ο κώδικας C, για δική μου διευκόλυνση στην δοκιμή διαφόρων εισόδων, περιέχει και ρουτίνες διαβάσματος αρχείων, πέρα από την επίλυση των προβλημάτων. Τα περισσότερα προγράμματα χρησιμοποιούνται σε συνδυασμό με κάποιο από τα scripts που αναλύονται παρακάτω. Αυτό προσθέτει μεν πολυπλοκότητα, αλλά κάνει τα προγράμματα πιο ευέλικτα. Σε κάθε δραστηριότητα εξηγώ πως πρέπει να τρέξουμε το πρόγραμμα.

1.2 Scripts

Υπάρχουν τα εξής 3 βοηθητικά scripts:

- `atoh`: Μετατρέπει ένα string από ASCII σε Hex.
- `htoa`: Μετατρέπει ένα string από Hex σε ASCII.
- `tests`: Αναδεικνύει/αυτοματοποιεί την λειτουργία όλων των προγραμμάτων.

1.3 Αρχεία δεδομένων

Στο directory `src/dat` βρίσκονται αρχεία δεδομένων που χρησιμοποιούνται από τα προγράμματα. 'Ολα τα αρχεία ακολουθούν την ονομασία `<program>.in` όπου `program` το όνομα του προγράμματος που το χρησιμοποιεί. Προκειμένου να αποφευχθεί τυχόν περιττή πολυπλοκότητα, τα προγράμματα δεν κάνουν ελέγχους εγκυρότητας των αρχείων εισόδου.

1.4 Makefile

Το `Makefile` διαθέτει τις παρακάτω επιλογές:

- `make`: Κάνει compile όλα τα προγράμματα.
- `make clean`: Σβήνει τα εκτελέσιμα αρχεία που έχουν παραχθεί.

2 Δραστηριότητα 1: Δημιουργία ιδιωτικού κλειδιού

Κώδικας: `priv.c`
Χρήση: `priv [-v] input`

Για να δημιουργήσουμε ένα ιδιωτικό κλειδί RSA, πρέπει αρχικά να υπολογίσουμε την συνάρτηση:

$$\phi(n) = (p-1)(q-1)$$

Στην συνέχεια, θα υπολογίσουμε την εξίσωση:

$$e \cdot d \mod \phi(n) = 1$$

Λύνοντας ως προς d. Ο υπολογισμός του ιδιωτικού κλειδιού μέσα στο πρόγραμμα γίνεται με τις εξής εντολές:

```
...
BN_dec2bn(&one, "1");
BN_sub(foo, p, one);
BN_sub(bar, q, one);
BN_mul(phi, foo, bar, ctx);
BN_mod_inverse(d, e, phi, ctx);
```

Το πρόγραμμα, όταν το τρέξουμε απλώς με το αρχείο εισόδου, τυπώνει το ιδιωτικό κλειδί. Αν του δώσουμε και την επιλογή -v, τυπώνει αναλυτικά τα e, n και d. Η επιλογή αυτή είναι χρήσιμη για την παραγωγή του αρχείου εισόδου που χρησιμοποιείται για την κρυπτογράφηση μηνυμάτων.

Ενδεικτικά τρεξίματα:

```
$ ./priv dat/priv.in
63F67E805D8DEB0B4182C57C3DC24F3C1350CF182E8ABF85FD24062A3BC7F2EB

$ ./priv -v dat/priv.in
e: 0D88C3
n: 71D9BBC5C01F9B50DDFE5F2EC331FAB21081009D014E9615C277670C61591ECF
d: 63F67E805D8DEB0B4182C57C3DC24F3C1350CF182E8ABF85FD24062A3BC7F2EB
```

3 Δραστηριότητα 2: Κρυπτογράφηση μηνύματος

Κώδικας: `encrypt.c`
Χρήση: `./atoh 'msg' | encrypt input`

Η κρυπτογράφηση ενός μηνύματος γίνεται με τον τύπο:

$$C = P^e \mod n$$

Και η αποκρυπτογράφηση του:

$$P = C^d \mod n$$

Η συνάρτηση OpenSSL για την πράξη αυτή είναι η `BN_mod_exp()`. Οι παρακάτω εντολές στον κώδικα εκτελούν την (απο)κρυπτογράφηση:

```
...
/* Encrypt message */
BN_mod_exp(encrstr, str, e, n, ctx);
/* Decrypt message */
BN_mod_exp(decrstr, encrstr, d, n, ctx);
```

Παρακάτω φαίνεται ένα ενδεικτικό τρέξιμο. Το μήνυμα μετατρέπεται σε Hex με την χρήση του `atoh` script. Αξίζει να σημειωθεί ότι στην υλοποίησή μου, τα e, n και d υπολογίζονται από το `priv.c` και χρησιμοποιούνται κατευθείαν από το `encrypt.c` ώστε να αποφευχθεί η επανάληψη κώδικα, εξ'ου και η χρήση του `priv` στην αρχή:

```
$ ./priv -v dat/priv.in | awk '{print $2}' > dat/encrypt.in
$ ./atoh 'Christos Margiolis' | ./encrypt dat/encrypt.in
received: 69726843736F747372614D206C6F69677369
encrypted: 192FDF7BCA2F402253E344F1A25476D4276523750DED4BE6FC058F179219BA6D
decrypted: 69726843736F747372614D206C6F69677369
```

4 Δραστηριότητα 3: Αποκρυπτογράφηση μηνύματος

Κώδικας: `decrypt.c`
Χρήση: `decrypt input | htoa`

'Οπως και στην δραστηριότητα 3, χρησιμοποιείται ο ίδιος τύπος για την αποκρυπτογράφηση ενός μηνύματος. Η υλοποίηση είναι η ίδια, με την διαφορά ότι τα δεδομένα εισόδου είναι διαφορετικά. Το πρόγραμμα τυπώνει το

αποκρυπτογραφημένο μήνυμα σε Hex, οπότε διοχετεύουμε την έξοδό του στο `htoa` script ώστε να μετατραπεί σε ASCII:

```
$ ./decrypt dat/decrypt.in | ./htoa
INFOSEC Spring 2022
```

5 Δραστηριότητα 4: Υπογραφή μηνύματος

Κώδικας: `sign.c`
Χρήση: `atoh 'msg' | sign input`

Η υπογραφή ενός μηνύματος γίνεται με τον τύπο:

$$S = H(P)^d \mod n$$

Στον κώδικα, υλοποίηση είναι γίνεται ως εξής:

```
...
BN_mod_exp(sign, str, d, n, ctx);
```

Στο παρακάτω ενδεικτικό τρέξιμο, παρατηρούμε ότι μία πολύ μικρή αλλαγή στο μήνυμα θα παράξει τελείως διαφορετική υπογραφή, οπότε είμαστε και σίγουροι ότι τα μηνύματα δεν ήτανε ίδια:

```
$ ./atoh 'This is a message' | ./sign dat/decrypt.in
96798DC95A5ECBBEF35D6D68588157CA2DAD163B45453B4B6D80FCC3BE15E8ED

$ ./atoh 'This iz a message' | ./sign dat/decrypt.in
D7562984FFA684E2850A2763F4DD1DA045EFD2DB4CFBF6F40E6579F3E3AD9536
```

6 Δραστηριότητα 5: Επαλήθευση υπογραφής

Η επαλήθευση της υπογραφής γίνεται με τον τύπο:

$$Digest = S^e \mod n$$

Στον κώδικα, υλοποίηση είναι γίνεται ως εξής:

```
...
BN_mod_exp(str, sign, e, n, ctx);
```

6.1 Περίπτωση Α

Αρχικά θα δώσουμε ως είσοδο την έγκυρη υπογραφή:

```
$ ./verify dat/verify1_cor.in
e: 010001
n: AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115
sign: 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F
str: 4C61756E63682061206D697373696C652E

$ ./verify dat/verify1_cor.in | tail -1 | awk '{print $2}' | ./htoa
Launch a missile.
```

'Οταν αλλάξουμε το τελευταίο byte της υπογραφής, παρατηρούμε ότι η επαλήθευση δεν είναι έγκυρη:

```
$ ./verify dat/verify1_inc.in
e: 010001
n: AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115
sign: 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F
str: 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294

$ ./verify dat/verify1_cor.in | tail -1 | awk '{print $2}' | ./htoa
??,O?c??rm=f?:N?????
```

6.2 Περίπτωση Β

Από το παρακάτω τρέξιμο, βλέπουμε ότι η υπόγραφη είναι πράγματι της Alice:

```
$ ./verify dat/verify2.in
e: 010001
n: DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
sign: DB3F7CDB93483FC1E70E4EACA650E3C6505A3E5F49EA6EDF3E95E9A7C6C7A320
str: 506C656173652074726E73666572206D652024323030302E416C6963652E

$ ./verify dat/verify2.in | tail -1 | awk '{print $2}' | ./htoa
Please transfer me $2000.Alice.
```

7 Δραστηριότητα 6: Μη-αυτόματη επαλήθευση πιστοποιητικού X.509

Κατεβάζουμε το πιστοποιητικό της ιστοσελίδας margiolis.net:

```
$ openssl s_client -connect margiolis.net:443 -showcerts \
        </dev/null 2>/dev/null | openssl x509 -outform pem > dat/c0.pem
```

Εξάγουμε το e:

```
$ openssl x509 -in dat/c0.pem -text -noout | grep 'Exponent' |
        awk '{print $3}' | sed 's/(//;s/)//;s/0x//' > dat/cert.in
```

Εξάγουμε το n:

```
$ openssl x509 -in dat/c0.pem -noout -modulus |
        sed 's/Modulus=//' >> dat/cert.in
```

Εξάγουμε την υπογραφή:

```
$ openssl x509 -in dat/c0.pem -text -noout \
        -certopt ca_default -certopt no_validity \
        -certopt no_serial -certopt no_subject \
        -certopt no_extensions -certopt no_signame |
        sed 1d | tr -d '[:space:]:' | sha256 >> dat/cert.in
```

Τέλος, επαληθεύουμε το πιστοποιητικό (το output είναι πολύ μεγάλο για να συμπεριληφθεί ολόκληρο):

```
$ ./verify dat/cert.in
e: 010001
n: B8CF8F.........1AE7F0DE351B
sign: E8230B..........AC59DF719
str: 46F35C99.........5034620EF8149AE
```