Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Υπολογιστών

Εργαστήριο Ασφάλειας στην Τεχνολογία της Πληροφορίας – Εργασία 2

Χρήστος Μαργιώλης – 19390133

Απρίλιος 2023

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

Πέρα από τον κώδικα έγραψα και τα παρακάτω 2 scripts.

1.2.1 `atoh`: Μετατροπή από ASCII σε Hex

```
#!/bin/sh

echo -n ${1} | hexdump -e '"%02X"' | sed 's/$/\n/'
```

1.2.2 `htoa`: Μετατροπή από Hex σε ASCII

```
#!/bin/sh

while read hex; do
        echo "16i ${hex} P" | dc
done
echo
```

1.3 Makefile

Τα προγράμματα μεταγλωττίζονται όλα αυτόματα μέσω του παρακάτω `Makefile`.

Το `Makefile` διαθέτει τις παρακάτω επιλογές:

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

Ο κώδικας του:

```
TARGS = priv \
        encrypt \
        decrypt \
        sign \
        verify

SCRIPTS = atoh \
          htoa

CC = cc
CFLAGS = -std=c99 -pedantic -Wall -Os -Iinclude
LIBS = -Llib -lcrypto

all:
```

```
        for targ in ${TARGS} ; do \
                ${CC} $${targ}.c ${LIBS} -o $${targ} ; \
        done
        chmod +x ${SCRIPTS}

clean:
        rm -f ${TARGS} *.o *.core
```

1.4 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.

2 Δραστηριότητα 1: Δημιουργία ιδιωτικού κλειδιού

2.1 Επεξήγηση υλοποίησης

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);
```

2.2 Εκτέλεση προγράμματος

Χρήση: `priv [-v] input`

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:

```
christos@pleb$ cat dat/priv.in
F7E75FDC469067FFDC4E847C51F452DF
E85CED54AF57E53E092113E62F436F4F
0D88C3
christos@pleb$ ./priv dat/priv.in
3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
christos@pleb$ ./priv -v dat/priv.in
e: 0D88C3
n: E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
d: 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
christos@pleb$
```

2.3 Πηγαίος κώδικας: priv.c

```
#include <err.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include <openssl/bn.h>

static char *argv0;

static const char *
read_line(FILE *fp)
{
        char buf[2048];

        if (fgets(buf, sizeof(buf), fp) == NULL)
                err(1, "fgets");
        return (strdup(buf));
}

static void
printbn(char *str, BIGNUM *bn)
{
        char *s;

        s = BN_bn2hex(bn);
        printf("%s%s\n", str, s);
        OPENSSL_free(s);
}

static void
usage(void)
{
```

```
        fprintf(stderr, "usage: %s [-v] input\n", argv0);
        exit(1);
}

int
main(int argc, char *argv[])
{
        BN_CTX *ctx;
        BIGNUM *p, *q, *e, *n, *d;
        BIGNUM *phi, *one, *foo, *bar;
        FILE *fp;
        int verbose = 0;
        char ch;

        argv0 = *argv;

        while ((ch = getopt(argc, argv, "v")) != -1) {
                switch (ch) {
                case 'v':
                        verbose = 1;
                        break;
                case '?': /* FALLTHROUGH */
                default:
                        usage();
                }
        }
        argc -= optind;
        argv += optind;

        if (!argc)
                usage();
        if ((fp = fopen(*argv, "r")) == NULL)
                err(1, "fopen(%s)", *argv);

        ctx = BN_CTX_new();
        p = BN_new();
        q = BN_new();
        e = BN_new();
        n = BN_new();
        d = BN_new();

        phi = BN_new();
        one = BN_new();
        foo = BN_new();
        bar = BN_new();

        /* We assume the file has at least 3 lines */
        BN_hex2bn(&p, read_line(fp));
        BN_hex2bn(&q, read_line(fp));
        BN_hex2bn(&e, read_line(fp));
```

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

        /*
         * Calculate private key:
         * 1. phi(n) = (p-1) * (q-1)
         * 2. (e * d mod phi(n) = 1), solve for d
         */
        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);

        if (verbose) {
                printbn("e: ", e);
                printbn("n: ", n);
                printbn("d: ", d);
        } else
                printbn("", d);

        fclose(fp);
        OPENSSL_free(p);
        OPENSSL_free(q);
        OPENSSL_free(e);
        OPENSSL_free(n);
        OPENSSL_free(q);
        OPENSSL_free(phi);
        OPENSSL_free(one);
        OPENSSL_free(foo);
        OPENSSL_free(bar);
        OPENSSL_free(ctx);

        return (0);
}
```

3 Δραστηριότητα 2: Κρυπτογράφηση μηνύματος

3.1 Επεξήγηση υλοποίησης

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);
```

3.2 Εκτέλεση προγράμματος

Χρήση: `./atoh 'msg' | encrypt input`

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:

```
christos@pleb$ cat atoh
#!/bin/sh

echo -n ${1} | hexdump -e '"%02X"' | sed 's/$/\n/'
christos@pleb$ ./priv -v dat/priv.in | awk '{print $2}' > dat/encrypt.in
christos@pleb$ ./atoh 'Christos Margiolis' | ./encrypt dat/encrypt.in
received: 69726843736F747372614D206C6F69677369
encrypted: 1E8FF0654041076328165697C0D317A4275BE282908D3F163C37A3C348E03F62
decrypted: 69726843736F747372614D206C6F69677369
```

3.3 Πηγαίος κώδικας: `encrypt.c`

```
#include <err.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>

#include <openssl/bn.h>

static const char *
read_line(FILE *fp)
{
        char buf[2048];

        if (fgets(buf, sizeof(buf), fp) == NULL)
                err(1, "fgets");
        return (strdup(buf));
}

static void
printbn(char *str, BIGNUM *bn)
{
        char *s;

        s = BN_bn2hex(bn);
```

```
        printf("%s%s\n", str, s);
        OPENSSL_free(s);
}

int
main(int argc, char *argv[])
{
        BN_CTX *ctx;
        BIGNUM *str, *encrstr, *decrstr;
        BIGNUM *e, *n, *d;
        FILE *fp;
        int len = 0;
        char buf[2048];

        if (argc < 2) {
                fprintf(stderr, "usage: %s input\n", *argv);
                return (-1);
        }
        if ((fp = fopen(argv[1], "r")) == NULL)
                err(1, "fopen(%s)", argv[1]);

        /* Read string from stdin */
        while (read(STDIN_FILENO, &buf[len++], 1) > 0)
                ;
        buf[--len] = '\0';

        ctx = BN_CTX_new();
        str = BN_new();
        encrstr = BN_new();
        decrstr = BN_new();
        e = BN_new();
        n = BN_new();
        d = BN_new();

        BN_hex2bn(&str, buf);
        /*
         * Assumes input from file produced by `priv -v`, so that we
         * avoid duplicating code to recalculate p, q, e, n, d.
         */
        BN_hex2bn(&e, read_line(fp));
        BN_hex2bn(&n, read_line(fp));
        BN_hex2bn(&d, read_line(fp));

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

        printbn("received: ", str);
        printbn("encrypted: ", encrstr);
```

```
        printbn("decrypted: ", decrstr);

        fclose(fp);
        OPENSSL_free(e);
        OPENSSL_free(n);
        OPENSSL_free(d);
        OPENSSL_free(ctx);

        return (0);
}
```

4 Δραστηριότητα 3: Αποκρυπτογράφηση μηνύματος

4.1 Επεξήγηση υλοποίησης

'Οπως και στην δραστηριότητα 3, χρησιμοποιείται ο ίδιος τύπος για την αποκρυπτογράφηση ενός μηνύματος.

4.2 Εκτέλεση προγράμματος

Χρήση: `decrypt input | htoa`

Στο αρχείο εισόδου θα χρησιμοποιήσουμε τα ίδια κλειδιά με αυτά της άσκησης 3, συν ότι θα προσθέσουμε και το επιπλέον κρυπτογράφημα που δίνεται στην εκφώνηση της άσκησης. Το πρόγραμμα τυπώνει το αποκρυπτογραφημένο μήνυμα σε Hex, οπότε διοχετεύουμε την έξοδό του στο **htoa** script ώστε να μετατραπεί σε ASCII:

```
christos@pleb$ ./priv -v dat/priv.in | awk '{print $2}' > dat/decrypt.in
christos@pleb$ echo '7074A77B724869F59EBA8B790CEAA6A33B2E268A00BAF124BA7C3A6BDE690F36' >> dat/decrypt.in
christos@pleb$ ./decrypt dat/decrypt.in | ./htoa
Information Security Lab 2023
```

4.3 Πηγαίος κώδικας: `decrypt.c`

```
#include <err.h>
#include <stdio.h>
#include <string.h>

#include <openssl/bn.h>

static const char *
read_line(FILE *fp)
{
        char buf[2048];

        if (fgets(buf, sizeof(buf), fp) == NULL)
                err(1, "fgets");
        return (strdup(buf));
}

static void
```

```
printbn(char *str, BIGNUM *bn)
{
        char *s;

        s = BN_bn2hex(bn);
        printf("%s%s\n", str, s);
        OPENSSL_free(s);
}

int
main(int argc, char *argv[])
{
        BN_CTX *ctx;
        BIGNUM *e, *n, *d, *c, *decrstr;
        FILE *fp;

        if (argc < 2) {
                fprintf(stderr, "usage: %s input\n", *argv);
                return (-1);
        }
        if ((fp = fopen(argv[1], "r")) == NULL)
                err(1, "fopen(%s)", argv[1]);

        ctx = BN_CTX_new();
        e = BN_new();
        n = BN_new();
        d = BN_new();
        c = BN_new();
        decrstr = BN_new();

        BN_hex2bn(&e, read_line(fp));
        BN_hex2bn(&n, read_line(fp));
        BN_hex2bn(&d, read_line(fp));
        BN_hex2bn(&c, read_line(fp));

        BN_mod_exp(decrstr, c, d, n, ctx);
        printbn("", decrstr);

        fclose(fp);
        OPENSSL_free(e);
        OPENSSL_free(n);
        OPENSSL_free(d);
        OPENSSL_free(c);
        OPENSSL_free(decrstr);
        OPENSSL_free(ctx);

        return (0);
}
```

5 Δραστηριότητα 4: Υπογραφή μηνύματος

5.1 Επεξήγηση υλοποίησης

Η υπογραφή ενός μηνύματος γίνεται με τον τύπο:

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

Στον κώδικα, η υλοποίηση γίνεται ως εξής:

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

5.2 Εκτέλεση προγράμματος

Χρήση: `atoh 'msg' | sign input`

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:

```
christos@pleb$ ./atoh 'This is a message' | ./sign dat/decrypt.in
709A1736BE38AD833047AE8495F67D1EC63635527135731AD5CDB113E1BCF2F8
christos@pleb$ ./atoh 'This iz a message' | ./sign dat/decrypt.in
2F881F3999EEB3142C29109E6AE595A125D1A7056360C9DA77CAAA578993AB02
```

5.3 Πηγαίος κώδικας: `sign.c`

```
#include <err.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include <openssl/bn.h>

static const char *
read_line(FILE *fp)
{
        char buf[2048];

        if (fgets(buf, sizeof(buf), fp) == NULL)
                err(1, "fgets");
        return (strdup(buf));
}

static void
printbn(char *str, BIGNUM *bn)
{
        char *s;
```

```
        s = BN_bn2hex(bn);
        printf("%s%s\n", str, s);
        OPENSSL_free(s);
}

int
main(int argc, char *argv[])
{
        BN_CTX *ctx;
        BIGNUM *e, *n, *d, *c, *str, *sign;
        FILE *fp;
        int len = 0;
        char buf[2048];

        if (argc < 2) {
                fprintf(stderr, "usage: %s input\n", *argv);
                return (-1);
        }
        if ((fp = fopen(argv[1], "r")) == NULL)
                err(1, "fopen(%s)", argv[1]);

        /* Read string from stdin */
        while (read(STDIN_FILENO, &buf[len++], 1) > 0)
                ;
        buf[--len] = '\0';

        ctx = BN_CTX_new();
        e = BN_new();
        n = BN_new();
        d = BN_new();
        c = BN_new();
        str = BN_new();
        sign = BN_new();

        BN_hex2bn(&e, read_line(fp));
        BN_hex2bn(&n, read_line(fp));
        BN_hex2bn(&d, read_line(fp));

        BN_hex2bn(&str, buf);
        BN_mod_exp(sign, str, d, n, ctx);
        printbn("", sign);

        fclose(fp);
        OPENSSL_free(e);
        OPENSSL_free(n);
        OPENSSL_free(d);
        OPENSSL_free(c);
        OPENSSL_free(str);
        OPENSSL_free(sign);
```

```
        OPENSSL_free(ctx);

        return (0);
}
```

6 Δραστηριότητα 5: Επαλήθευση υπογραφής

6.1 Επεξήγηση υλοποίησης

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);
```

6.2 Εκτέλεση προγράμματος

Χρήση: `verify input`

6.3 Περίπτωση Α

Αρχικά θα δώσουμε ως είσοδο την έγκυρη υπογραφή:

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

```
christos@pleb$ sed '$ s/.$/2/' dat/verify1_cor.in > dat/verify1_inc.in
christos@pleb$ ./verify dat/verify1_inc.in
e: 010001
n: AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115
sign: 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB68022
str: 488455F9D7F2E7BD0AF67736088DF289F44DB0B656F016BACBE4E3D445E60990
christos@pleb$ ./verify dat/verify1_inc.in | tail -1 | awk '{print $2}' | ./htoa
HU
wMVE
```

6.4 Περίπτωση B

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

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

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

Για την συγκεκριμένη δραστηριότητα επέλεξα να παραθέσω τις εντολές σε μορφή text αντί για screenshots διότι τα output και οι ίδιες οι εντολές είναι πολύ μεγάλες για να φανούν καθαρά σε εικονα.

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: B8CF80904908D88.........1AE7F0DE351B
sign: EC3CF68F5F6..........D228F04C5E54BE1D
str: 6E7DBA8412AEB7CF.........5FE55D1059486304
```