ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ
ECE_Y215: ΔΙΑΔΙΚΑΣΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

3η Δραστηριότητα

Χρήση της Βιβλιοθήκης GMP(GNU MultiPrecision Library) για την Υλοποίηση του RSA Αλγορίθμου Κρυπτογραφίας

Εισαγωγή

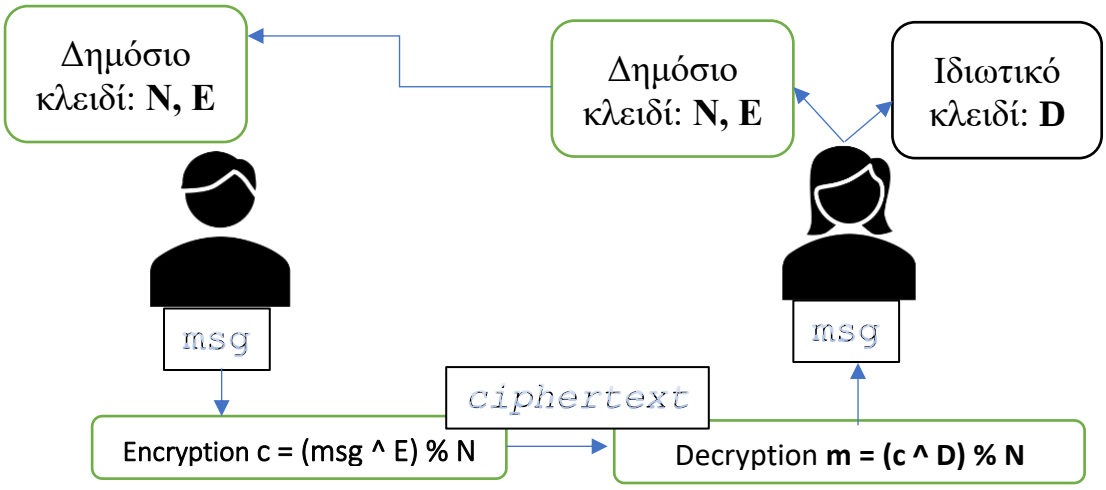
Σκοπός της παρούσας δραστηριότητας είναι η εξοικείωση με την βιβλιοθήκη GMP [1] με σκοπό τη δημιουργία ενός προγράμματος υλοποίησης του αλγόριθμού κρυπτογραφίας RSA [2].

Το πρόβλημα της ασφαλούς επικοινωνίας αποτελεί ένα διαχρονικό πρόβλημα στην επιστήμη των υπολογιστών. Οι αλγόριθμοι κρυπτογραφίας έχουν ως στόχο να μετασχηματίσουν το μήνυμα της επικοινωνίας έτσι ώστε αν υποκλαπεί να μην μπορεί να κατανοηθεί με εύκολο τρόπο. Το 1977 οι Ron **R**ivest, Adi **S**hamir και Leonard **A**dlerman πρότειναν μια μέθοδο ασύμμετρης κρυπτογραφίας με χρήση ενός δημοσίου και ιδιωτικού κλειδιού η οποία βασίζεται στη θεωρία αριθμών και ειδικότερα: α) στη συνάρτηση Φ του Euler [3] β) το μικρό θεώρημα του Fermat [4] γ) τον αλγόριθμο εύρεσης σχετικών πρώτων αριθμών του Ευκλείδη [5] και τέλος δ) το κινεζικό θεώρημα υπολοίπων [6].

Βάση των παραπάνω μπόρεσαν να υπολογίζουν τέτοιους αριθμούς E,N (*δημόσιο κλειδί*) και D (*ιδιωτικό κλειδί*) ώστε να μην είναι υπολογιστικά δυνατή (*με συμβατούς υπολογιστές όχι όμως και με κβαντικούς [7]*) η εύρεση του αριθμού D που αντιστοιχεί σε δημόσιο κλειδί μεγάλου μήκους (N) το οποίο συνήθως είναι 1024 ή 2048 bits.

Στο ***Σχήμα 1*** η Άννα δημιουργεί βάση του RSA αλγορίθμου το δημόσιο και ιδιωτικό της κλειδί. Στη συνέχεια δίνει δημοσίως μόνο το δημόσιο κλειδί ενώ το ιδιωτικό κλειδί το κρατά μυστικό. Η κωδικοποίηση των μηνυμάτων επιτυγχάνεται με τη χρήση του δημοσίου κλειδιού ενώ η αποκωδικοποίηση των κρυπτογραφημένων μηνυμάτων μπορεί να γίνει μόνο μέσω του ιδιωτικού της κλειδιού.

Σήμερα ο αλγόριθμος RSA χρησιμοποιείται καθημερινά ανά την υφήλιο για την κρυπτογραφία της πληροφορίας που μεταδίδεται στο διαδίκτυο ενώ αποτελεί παγκοσμίως τον πρώτο σε αριθμό υλοποιήσεων αλγόριθμο.



Σχήμα 1: Ασύμμετρη Κρυπτογραφία RSA με δημόσιο [N, E] και ιδιωτικό κλειδί [D].

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ
ECE_Y215: ΔΙΑΔΙΚΑΣΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Ζητούμενα

Δημιουργήστε ένα πρόγραμμα στη C με χρήση της βιβλιοθήκης GMP το οποίο θα κρυπτογραφεί βάση του αλγορίθμου RSA ένα μήνυμα που θα εισάγει ο χρήστης από το πληκτρολόγιο. Αφού ο χρήστης εισάγει το μήνυμα από το πληκτρολόγιο στη συνέχεια το πρόγραμμα θα ζητά τον αριθμό των bits βάση του οποίου θα υπολογίζει το δημόσιο [N, E] και ιδιωτικό κλειδί [D].

Παραδοτέα

Η δραστηριότητα περιλαμβάνει τρεις εκδόσεις. Στο Παράρτημα δίνεται ένας βασικός σκελετός κώδικα για κάθε έκδοση, ο οποίος περιλαμβάνει τις δηλώσεις των συναρτήσεων τις οποίες πρέπει να υλοποιήσετε. Ωστόσο, είστε ελεύθεροι να δημιουργήσετε ανεξάρτητα από τον ενδεικτικό κώδικα όποιο κώδικα εσείς επιθυμείτε.

Έκδοση 1: Προσδιορισμός του μεγέθους του κλειδιού κρυπτογράφησης [*N*]. Υπολογισμός του αριθμού N βάση των πρώτων αριθμών [*p,q*]].

Ο σκοπός της πρώτης έκδοσης είναι να δημιουργήσετε το πρώτο μέρος του δημοσίου κλειδιού RSA που είναι ο αριθμός (N). Για να δημιουργήσουμε τυχαίους αριθμούς συγκεκριμένου αριθμού bit χρησιμοποιούμε την συνάρτηση *mpz_urandomb* που δίνεται μέσω της γεννήτριας ψευδοτυχαίων αριθμών της GMP. Στη συνέχεια για να μπορέσουμε να βρούμε αν ένας ακέραιος αριθμός είναι πρώτος αριθμός (prime number) χρησιμοποιούμε τη συνάρτηση *mpz_probab_prime_p.* Προσπαθήσετε να τρέξετε το πρόγραμμα με μεταβλητό μήκος κλειδιού [*2 bit*, *64 bit*, *128 bit*, *256 bit*, *512 bit*, *1024* bit, *2048* bit].

Στην **Εικόνα 1** του Παραρτήματος, υπάρχει ενδεικτικός πηγαίος κώδικας της πρώτης έκδοσης. Στον πηγαίο κώδικα φαίνονται οι δηλώσεις των συναρτήσεων χωρίς την υλοποίησή τους[1]. Βασικές συναρτήσεις της πρώτης έκδοσης είναι:

- void **init_gmp_rand**(*gmp_randstate_t *stat*);
- void **generate_prime**(*mpz_t* p, gmp_randstate_t *stat, int *bit_length*);
- void **calulateN**(*mpz_t* n, mpz_t* p, mpz_t* q*);

Έκδοση 2: Υπολογισμός των αριθμών E και D

Ο σκοπός της δεύτερης έκδοσης είναι ο υπολογισμός του δεύτερου αριθμού του δημοσίου κλειδιού E καθώς και του αριθμού D το οποίο είναι το ιδιωτικό κλειδί . Για το σκοπό αυτό βρίσκουμε τον αριθμό Φ(N) του Euler. Επειδή Φ(N)=Φ(p)*Φ(q) και γνωρίζοντας ότι οι αριθμοί p και q είναι πρώτοι αριθμοί ισχύει: Φ(N)= (p-1)*(q-1).

Στη συνέχεια υπολογίζουμε έναν ακέραιο E στο διάστημα (1,Φ(N)) με κριτήριο τέτοιο ώστε οι αριθμοί E και Φ(N) να είναι σχετικά πρώτοι (Μ.Κ.Δ.=1). Μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο του Ευκλείδη για να διαπιστώσουμε αν δυο αριθμοί είναι σχετικά πρώτοι. Ο αλγόριθμος του Ευκλείδη υλοποιείται από τη βιβλιοθήκη GMP μέσω της συνάρτησης *mpz_gcd.* Τέλος υπολογίζουμε τον αριθμό D βάση του μικρού θεωρήματος του Fermat [5] και το κινεζικό θεώρημα υπολοίπων [6]. Η συνάρτηση *mpz_invert* της GMP υπολογίζει τον αριθμό D βάση των E και Φ(N).

Στην **Εικόνα 2** του Παραρτήματος υπάρχει ενδεικτικός πηγαίος κώδικας της δεύτερης έκδοσης. Στον πηγαίο κώδικα φαίνονται οι δηλώσεις των συναρτήσεων χωρίς την υλοποίησή τους[2]. Βασικές συναρτήσεις της δεύτερης έκδοσης είναι:

void **generate_eulers_phi**(mpz_t* p, mpz_t* q, mpz_t* e);
void **generate_rsa_e**(mpz_t* phi, mpz_t* e);
void **generate_rsa_d**(mpz_t* d, mpz_t* phi, mpz_t* e);

---

[1] *Σημείωση: Η ονομασία των συναρτήσεων καθώς και τα ορίσματα είναι ενδεικτικά. Μπορείτε να επιλέξετε οποιοδήποτε ονομασία συναρτήσεων επιθυμείτε.*

Έκδοση 3: Κωδικοποίησης και αποκωδικοποίησης μηνύματος μέσω του δημοσίου και ιδιωτικού κλειδιού που δημιουργήσαμε στις εκδόσεις 1 και 2

Ο σκοπός της τρίτης έκδοσης είναι να χρησιμοποιήσουμε τα κλειδιά μας για να κωδικοποιήσουμε και να αποκωδικοποιήσουμε κάποιο μήνυμά. Για τον υπολογισμό του **c = (msg ^ E) % N** μπορείτε να χρησιμοποιήσετε την συνάρτηση της GMP *mpz_powm(*c, msg, *e, *n).*

Αντίστοιχα για την αποκωδικοποίηση του κρυπτογραφημένου μηνύματος **m = (c ^ D) % N** μπορείτε να χρησιμοποιήσετε αντίστοιχα την mpz_powm(*m, *c, *d, *n);

Στην **Εικόνα 3** του Παραρτήματος υπάρχει ενδεικτικός πηγαίος κώδικας της τρίτης έκδοσης. Στον πηγαίο κώδικα φαίνονται οι δηλώσεις των συναρτήσεων χωρίς την υλοποίησή τους[3]. Βασικές συναρτήσεις της τρίτης έκδοσης είναι:

- void **rsa_encryption** (*mpz_t* c, mpz_t* e, mpz_t* n, char msg[]*);
- void **rsa_decryption** (*mpz_t* m, mpz_t* c, mpz_t* d, mpz_t* n*);
- void **generate_msg** (*mpz_t* m, char msg[]*);

Καταγράψτε τις παρατηρήσεις σας σε μια μικρή έκθεση σχετικά με την ασφάλεια που μας παρέχει ο αλγόριθμος RSA. Η εργασία είναι προαιρετική και η εμπλοκή σας δεν σχετίζεται με βαθμολογικό αλλά μόνο με μαθησιακό κίνητρο. Ανεβάστε τις εκδόσεις του κώδικα σας μαζί με την αναφορά σας στο eclass σε μορφή συμπιεσμένου αρχείου και τίτλο τον αριθμό μητρώου σας (μέχρι τέλος Αυγούστου 2021) για να λάβετε σχόλια.

Αναφορές

[1] https://gmplib.org/
[2] https://en.wikipedia.org/wiki/RSA_(cryptosystem)
[3] https://en.wikipedia.org/wiki/Euler%27s_totient_function
[4] https://en.wikipedia.org/wiki/Fermat%27s_little_theorem
[5] https://en.wikipedia.org/wiki/Coprime_integers
[6] https://en.wikipedia.org/wiki/Chinese_remainder_theorem
[7] https://arxiv.org/pdf/1905.09749.pdf

Σημείωση: Η ονομασία των συναρτήσεων καθώς και τα ορίσματα είναι ενδεικτικά. Μπορείτε να επιλέξετε οποιοδήποτε ονομασία συναρτήσεων επιθυμείτε.

Παράρτημα
1η Έκδοση

```c
#include <stdio.h>
#include <stdlib.h>
#include "gmp.h"
#include <time.h>
#include<unistd.h>
void init_gmp_rand(gmp_randstate_t  *stat);
void generate_prime(mpz_t* p,gmp_randstate_t  *stat, int *bit_length);
int main()
{
    int prime_bits;
    printf("Enter RSA Encryption bits: ");
    scanf("%d",&prime_bits);

    gmp_randstate_t stat;
    init_gmp_rand(&stat);

    mpz_t p,q,n;
    generate_prime(&p,&stat, &prime_bits);
    generate_prime(&q,&stat, &prime_bits);
    gmp_printf("p = %Zd\n",p);
    gmp_printf("q = %Zd\n",q);

    mpz_init(n);
    mpz_mul(n,p,q);
    gmp_printf("N = %Zd\n",n);


   gmp_randclear(stat);
   mpz_clear(p);
   mpz_clear(q);
   mpz_clear(n);
   return 0;
}

void init_gmp_rand(gmp_randstate_t  *stat){
```

```
RSA - ENCRYPTION
RSA - Calculate p,q, N
Enter RSA Encryption bits: 1024
p = 5615760597909236528972066000006037302296000482073946811735810445639349159727
61324100318915865432579823084491517789662521211872105495187144474661044446180316
```

Εικόνα 1. Ενδεικτικός πηγαίος κώδικας και παράδειγμα εκτέλεσης της πρώτης έκδοσης.

2η Έκδοση

```c
#include <time.h>
#include<unistd.h>
void init_gmp_rand(gmp_randstate_t  *stat);
void rsa_first_step(mpz_t* p,mpz_t* q,mpz_t* n, gmp_randstate_t  *stat, int *bit_length);
void generate_prime(mpz_t* p,gmp_randstate_t  *stat, int *bit_length);

void generate_eulers_phi(mpz_t* p, mpz_t* q, mpz_t* e);
void generate_rsa_e(mpz_t* phi, mpz_t* e);
void generate_rsa_d(mpz_t* d, mpz_t* phi, mpz_t* e);
int main()
{
    int prime_bits;
    printf("Enter RSA Encryption bits: ");
    scanf("%d",&prime_bits);

    gmp_randstate_t stat;
    init_gmp_rand(&stat);

    mpz_t p,q,n;
    rsa_first_step(&p,&q,&n,&stat, &prime_bits);


    mpz_t phi,temp1, temp2;
    mpz_init(phi);
    mpz_init(temp1);
    mpz_init(temp2);

    mpz_sub_ui(temp1, q, 1);
    mpz_sub_ui(temp2, p, 1);
    mpz_mul(phi, temp1, temp2);
    gmp_printf("\nphi = %Zd\n",phi);
```

```
Enter RSA Encryption bits: 512
p = 19798978527421175382905821667342466536693943510907890668431607226270659158410077001705370680328159019625788781709386435652373520637250670068942702932829692
q = 64322474127408331767901173385122106542621539522099047144153847845606687559629699643085785459183397486059449484543888543129587273268361548733447160200329
N = 12735192840791616655036282912741466811667857231567691119311407439621568282450050710581344618109129730223987209418735935804089562043905784931375627137621626974411033320322504937945763690972290568819675586895450864602528864016990837093573762705391559920625402598258938513496921313687965047818089496300112389680 1

phi = 12735192840791616655036282912741466811667857231567691119311407439621568282450050710581344618109129730223987209418735935804089562043905784931375627137621626261331965067720274334298758131663265597756648452568260727387479781452435236563631037813431062107206954577344261729652437104769940313484752339335283670413504

e = 10929522261421456837192603892228657934008003078780348837852777373649775559019147464393608883310574892929870245584569379751095226525416355785212350966630690138428518982302597662634676488361157820743580745726782884351284866987023641676615926060238022859542051453820576078846986646756176954959283332065871648265

d = 95615778686039101348711871308716325457303997335170692589353640623903492489488012349436093435055690005828579445595747688902242244048601179792494348077303588766250861894070070248455823495239741258590795436297578550552177583781585396791783069749964431648496395685084536737595163809781321905379616438382291437 37
```

Εικόνα 2. Ενδεικτικός πηγαίος κώδικας και παράδειγμα εκτέλεσης δεύτερης έκδοσης.

3η Έκδοση

```c
#include <stdio.h>
#include <stdlib.h>
#include "gmp.h"
#include <time.h>
#include<unistd.h>
void init_gmp_rand(gmp_randstate_t  *stat);
void rsa_first_step(mpz_t* p,mpz_t* q,mpz_t* n, gmp_randstate_t  *stat, int *bit_length);
void generate_prime(mpz_t* p,gmp_randstate_t  *stat, int *bit_length);

void generate_eulers_phi(mpz_t* p, mpz_t* q, mpz_t* e);
void generate_rsa_e(mpz_t* phi, mpz_t* e);
void generate_rsa_d(mpz_t* d, mpz_t* phi, mpz_t* e);

void rsa_encryption (mpz_t* c, mpz_t* e, mpz_t* n,  char msg[] );
void rsa_decryption (mpz_t* m, mpz_t* c, mpz_t* d, mpz_t* n);
void generate_msg (mpz_t* m, char msg[]);



int main()
{
    int prime_bits;
    printf("Enter RSA Encryption bits: ");
    scanf("%d",&prime_bits);

    gmp_randstate_t stat;
    init_gmp_rand(&stat);

    mpz_t p,q,n;
    rsa_first_step(&p,&q,&n,&stat, &prime_bits);


    mpz_t phi,temp1, temp2;
    mpz_init(phi);
    mpz_init(temp1);
```

```
Enter RSA Encryption bits: 512
p = 19798978527421175382905821667342466536693943510907890668431607226270659158410077001705370680328159019625788781709386435652373520637250670068942702932829692
q = 64322474127408331767901173385122106542615395220990471441538478456066875596629699964308578545918339748605944948454388854312958727326836154873344716020032916
N = 12735192840791616655036282912741466811667857231567691119314074396215682824500507105813446181091297302239872094187359358040895620439057849313756271376216261331965067720274334298758131663265597756648452568260727387479781452435236563631037813431062107206954577344261729652437104769940313484752339335283670413504

e = 10929522261421456837192603892228657934008003078780348837852777373649775559019147464393608883310574892929870245584569379751095226525416355785212350966630690138428518982302597662634676488361157820743580745726782884351284866987023641676615926060238022859542051453820576078846986646756176954959283332065871648265

d = 95615778686039101348711871308716325457303997335170692589353640623903492489488012349436093435055690005828579445595747688902242244048601179792494348077303583766250861894070070248455823495239741258590795436297578550552177583781585396791783069749964431648496395685084536737595163809781321905379616438382291437370
```

```
RSA send = 250235687005031581218734895547219199167406216465662792252905318272797319268989868823118528426989775103737593773209914803963518751537061361818633525973310942738583731911352260862284950525358753263056041699994518275513331211453717185915943298282537877764046775664844192772452901881991229661016983311184548553960

Original Message: Hello RSA
```

Εικόνα 3. Ενδεικτικός πηγαίος κώδικας και παράδειγμα εκτέλεσης τρίτης έκδοσης.