

المدرسة العليا للتكنولوجيا - الداخلة
+ΣΙCΠ +οο.ΧΠΠο+ | +ΣΚΙ%Π%ΙΣ+ - ΛΛοΧΠο
ÉCOLE SUPÉRIEURE DE TECHNOLOGIE - DAKHLA



Le Compte Rendu du TD3/TP3 Cryptographie

Filière : Cybersécurité & Intelligence Artificielle

Module : Cryptographie

Chiffrement à clé publique : RSA

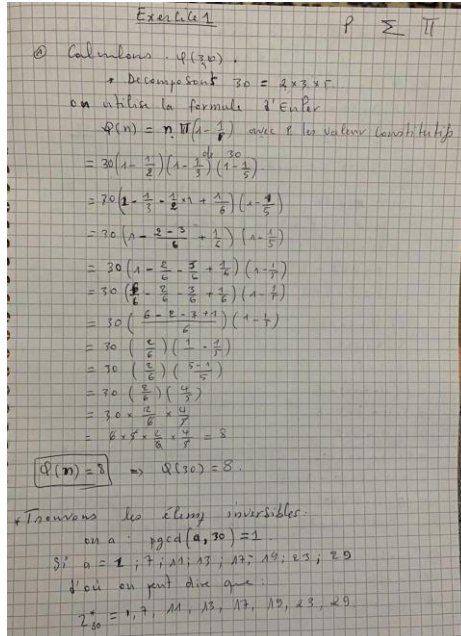
Année Universitaire : 2025 – 2026

Réalisé par : YAOGO Lucien

Encadré par : Pr. KHADIM

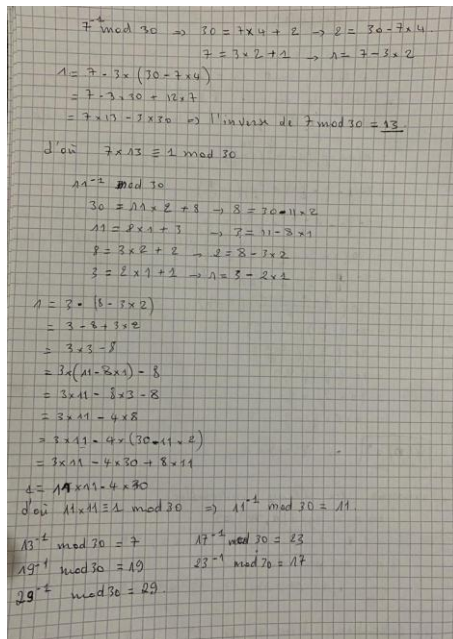
Exercice 1

1) Calcul de $\varphi(30)$



Nous avons calculé $\varphi(30)$ afin de déterminer le nombre d'éléments inversibles modulo 30, car la fonction d'Euler indique combien d'entiers sont premiers avec 30.

2) Détermination des éléments inversibles



Nous avons recherché les éléments de Z_{30} tels que $\text{pgcd}(a, 30) = 1$, car seuls ces éléments possèdent un inverse multiplicatif modulo 30.

Exercice 2

1) Chiffrement du message

Exo 2

$N = 187$ $e = 3$

1) Calculons le message $m = 15$ chiffré

$$x \equiv m^e \pmod{N}$$

$$\equiv 15^3 \pmod{187}$$

$$\equiv 15^{1+2} \pmod{187}$$

$$15^1 \equiv 15 \pmod{187}$$

$$15^2 \equiv 15 \times 15 \pmod{187}$$

$$\equiv 225 \pmod{187}$$

$$\equiv 38 \pmod{187}$$

$$15^3 \equiv 38 \times 15 \pmod{187}$$

$$15^3 \equiv 9 \pmod{187}$$

d'où $x \equiv 9 \pmod{187}$

d'où $c = 9$

Nous avons appliqué la formule $c = m^e \pmod{N}$ afin de transformer le message clair en message chiffré en utilisant la clé publique de Bob.

2) Retrouver la factorisation de N

$$\phi(N) = (p-1)(q-1) \quad N = pq$$

$$\begin{cases} (p-1)(q-1) = 160 \\ pq = 187 \end{cases} \Rightarrow \begin{cases} pq - p - q + 1 = 160 \\ pq = 187 \end{cases}$$

$$\Rightarrow \begin{cases} 187 - (p+q) + 1 = 160 \\ pq = 187 \end{cases} \Rightarrow \begin{cases} p+q = 28 \\ pq = 187 \end{cases}$$

en a: $187 = 11 \times 17$ et $28 = 11 + 17$

$p = 11$, $q = 17$

2) Calculons d c'est à dire la clé ^{privé} de Bob

$$d \equiv e^{-1} \pmod{\phi(N)} \quad \text{avec } e = 3$$

$$d \equiv 3^{-1} \pmod{160}$$

3) $3d \equiv 1 \pmod{160}$ Trouvons $3^{-1} \pmod{160}$

$$160 = 3 \times 53 + 1$$

$$\Rightarrow 1 = 160 - 3 \times 53$$

$$\Rightarrow 1 \equiv -3 \times 53 \pmod{160} \quad \text{donc l'inverse de } 3 \pmod{160}$$

d'où $d \equiv -53 \pmod{160}$ avec $-53 \equiv 107 \pmod{160}$

$d \equiv 107 \pmod{160}$ $d = 107$

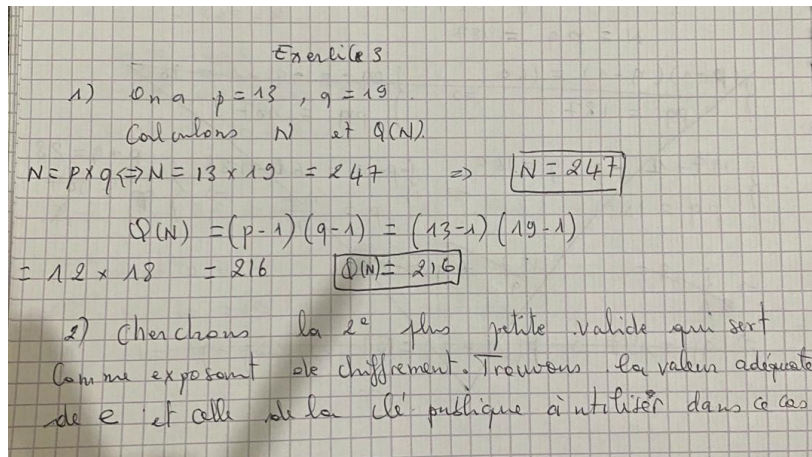
Nous avons utilisé la relation $\phi(N) = (p-1)(q-1)$ pour retrouver les facteurs premiers de N, car connaître $\phi(N)$ permet de casser la sécurité du système RSA.

Calcul de la clé privée

Nous avons calculé l'inverse modulaire de e modulo $\phi(N)$ afin d'obtenir la clé privée d , qui permet le déchiffrement des messages. D'où la clé privée de Bob est : **(107, 247)**

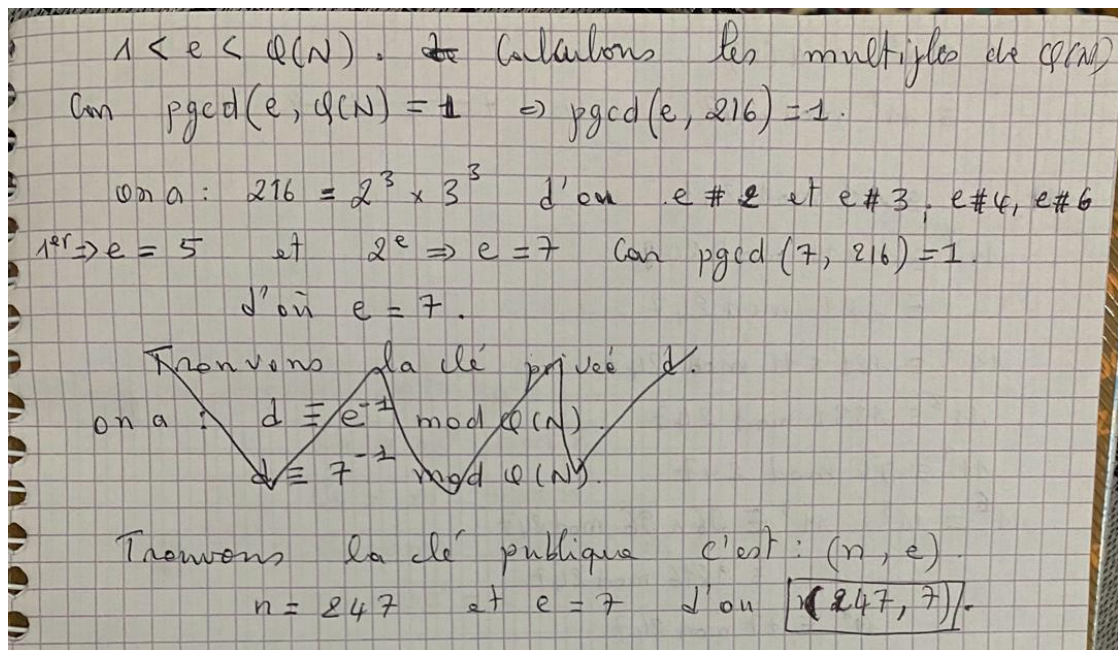
Exercice 3

1) Calcul de n



Nous avons multiplié p et q pour obtenir n , car le module n est utilisé dans les opérations de chiffrement et de déchiffrement.

2) Calcul de l'exposant e



Nous avons calculé $\varphi(n) = (p-1)(q-1)$ afin de pouvoir déterminer une clé privée valide.
 Nous avons choisi un entier e premier avec $\varphi(n)$ afin de garantir l'existence d'un inverse modulaire nécessaire au calcul de la clé privée.

3) Calcul de la clé privée d

3) Calculons la clé privée correspondante:
 on a: $d \equiv e^{-1} \pmod{\varphi(N)}$ avec $e = 7$ et $\varphi(N) = 216$.
 $\Leftrightarrow d \equiv 7^{-1} \pmod{216} \Rightarrow 7d \equiv 1 \pmod{216}$.

$7^{-1} \pmod{216}$:
 $216 = 7 \times 30 + 6$ $6 = 216 - 7 \times 30$
 $7 = 6 \times 1 + 1$ $\Rightarrow 1 = 7 - 6 \times 1$
 $\Rightarrow 1 = 7 - (216 - 7 \times 30)$
 $= 7 - 216 + 7 \times 30$
 $1 = 7 \times 31 - 216$
 $7 \times 31 \equiv 1 \pmod{216}$
 $7 \times 31 \equiv 1 \pmod{216}$.

d'où $d \equiv 31 \pmod{216} \Leftrightarrow \boxed{d = 31}$

clé publique: $(247, 7)$
 clé privée: $d = 31 \rightarrow (n, d) \Leftrightarrow (247, 31)$.

4) le message chiffré C correspondant $m = 11$.

Nous avons déterminé d tel que : $ed \equiv 1 \pmod{\varphi(n)}$ afin que le déchiffrement annule l'effet du chiffrement.

4) Chiffrement du message $m = 11$

$C \equiv m^e \pmod{n}$
 $\equiv 11^7 \pmod{247}$ avec $7 = 3 + 2 + 2$ $7 =$

$11^1 \equiv 11 \pmod{247}$
 $11^2 \equiv 11 \times 11 \pmod{247}$
 $\equiv 121 \pmod{247}$
 $11^3 \equiv 121 \times 11 \pmod{247}$
 $11^3 \equiv 1331 \pmod{247}$
 $11^3 \equiv 96 \pmod{247}$
 $11^6 \equiv 11^3 \times 11^3 \equiv 96 \times 96 \pmod{247}$
 $\equiv 9216 \pmod{247}$
 $11^6 \equiv 77 \pmod{247}$
 $11^7 \equiv 77 \times 11 \pmod{247}$
 $11^7 \equiv 847 \pmod{247}$
 $11^7 \equiv 106 \pmod{247}$ $\boxed{C = 106}$

Nous avons calculé $c = m^e \pmod{N}$ afin de produire le message chiffré à transmettre.

5) Déchiffrement du message $c = 23$

5) Trouvons le message m en clair du message chiffré $c = 23$.

$$m = c^d \pmod{n} \Rightarrow m = 23^{31} \pmod{247}$$

avec $31 = 16 + 8 + 4 + 2 + 1$.

$$\cancel{23^{31} = 23^{31}} \quad 23^{31} = 23^{1+2+4+8+16}$$

$$23^1 \equiv 23 \pmod{247}$$

$$23^2 \equiv 23 \times 23 \pmod{247}$$

$$\equiv 529 \pmod{247}$$

$$\equiv 35 \pmod{247}$$

$$23^4 \equiv 35 \times 35 \pmod{247}$$

$$\equiv 1225 \pmod{247}$$

$$\equiv 237 \pmod{247}$$

$$23^8 \equiv 237 \times 237 \pmod{247}$$

$$\equiv 56169 \equiv 100 \pmod{247}$$

$$23^8 \equiv 100 \pmod{247}$$

$$23^{16} \equiv 23^8 \times 23^8 \equiv 100 \times 100 \pmod{247}$$

$$23^{16} \equiv 10000 \pmod{247}$$

$$23^{16} \equiv 120 \pmod{247}$$

$$23^{31} \equiv 23^1 \times 23^2 \times 23^4 \times 23^8 \times 23^{16} \pmod{247}$$

$$23^{31} \equiv 23 \times 35 \times 237 \times 100 \times 120 \pmod{247}$$

Nous avons calculé $m = c^d \pmod{n}$ afin de retrouver le message clair à partir du message chiffré.

Oscar peut retrouver le message m

$$23^{31} \equiv 237 \pmod{247}$$

d'où $m = 237$.

6) Montrons que Oscar peut facilement découvrir le message m . on a : $e \neq e'$ avec $\text{pgcd}(e, e') = 1$

les données interceptés :

$$C_a = m^e \pmod{n}$$

$$C_b = m^{e'} \pmod{n}$$

d'après le théorème de Bézout on a :

$$ue + ve' = 1$$

on aura la relation suivante :

$$C_a^u \equiv (m^e)^u \pmod{n} \rightarrow \text{pour Alice}$$

$$C_b^v \equiv (m^{e'})^v \pmod{n} \rightarrow \text{pour Bob}$$

on a : $C_a^u \times C_b^v \equiv (m^e)^u \times (m^{e'})^v \pmod{n}$

$$\equiv m^{eu} \times m^{e'v} \pmod{n}$$

$$\equiv m^{eu+e'v} \pmod{n} \text{ or on a : } eu+e'v=1$$

$$\equiv m^1 \pmod{n}$$

d'où $m \equiv C_a^u \times C_b^v \pmod{n}$.

Nous avons montré que si deux utilisateurs partagent le même module n avec des exposants premiers entre eux, il est possible de combiner les deux cryptogrammes pour retrouver directement le message original en utilisant le théorème de Bézout.

Exercice 4

1) Fonction `premier (int n)`

```
#include <stdio.h>

// 1) Fonction pour tester si un nombre est premier
int premier(int n) {
    if (n <= 1) return 0;
    if (n == 2) return 1;
    if (n % 2 == 0) return 0;

    for (int i = 3; i * i <= n; i += 2) {
        if (n % i == 0) return 0;
    }
    return 1;
}

// La fonction principale pour le teste.
int main(){
    int a;
    printf("%d Entrez la valeur de a : ");
    scanf("%d", &a);
    int p = premier(a);

    if(p){
        printf("%d est un nombre premier ", a);
    }else{
        printf("%d n'est pas un nombre premier ", a);
    }

    return 0;
}
```

Nous avons écrit une fonction permettant de vérifier si un entier est premier, car RSA nécessite la génération de nombres premiers pour construire le module n .

2) Fonction `factorisation (int n)`

```
1 #include <stdio.h>
2 #include <string.h>
3
4
5 int premier(int n) {
6
7     if (n <= 1) return 0;
8     if (n == 2) return 1;
9     if (n % 2 == 0) return 0;
10
11     for (int i = 3; i * i <= n; i += 2) {
12         if (n % i == 0) return 0;
13     }
14     return 1;
15 }
16
17 int factorisation(int n) {
18
19     for (int i = 2; i <= n / 2; i++) {
20         if (n % i == 0) {
21             int j = n / i;
22             if (premier(i) && premier(j)) {
23                 return i;
24             }
25         }
26     }
27     return n;
28 }
29
30 // La fonction principale pour le teste
31 int main() {
32
33     int n;
34     printf("Entrez un entier n : ");
35     scanf("%d", &n);
36
37     int p = factorisation(n);
38     if (p != n) {
39         printf("n = %d x %d avec n premiers\n", p, n / p);
40     } else {
41         printf("n ne peut pas etre factorise en deux nombres premiers.\n");
42     }
43     return 0;
44 }
45
46
47 Line 30, Column 41
```

Nous avons implémenté une fonction de factorisation afin de comprendre que la sécurité de RSA repose sur la difficulté de décomposer n en produit de deux nombres premiers.

3) Fonction two_primes (int n, int prime [])

```
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 int premier(int n) {
7     if (n <= 1) return 0;
8     if (n == 2) return 1;
9     if (n % 2 == 0) return 0;
10    for (int i = 3; i * i <= n; i += 2) {
11        if (n % i == 0) return 0;
12    }
13    return 1;
14 }
15
16 // La fonction void two_primes()
17 void two_primes(int n, int prime[]){
18
19     int count = 0;
20     while(count < 2){
21         int x = rand() % n;
22         if (premier(x)){
23             prime[count] = x;
24             count++;
25         }
26     }
27 }
28
29 // La fonction principale pour le teste
30 int main() {
31
32     int n;
33     int c[2];
34
35     printf("Entrez un entier n : ");
36     scanf("%d", &n);
37
38     srand(time(NULL));
39
40     two_primes(n, c);
41
42     printf("Les deux nombres premiers sont : %d et %d\n", c[0], c[1]);
43
44     return 0;
45 }
```

Nous avons créé une fonction générant deux nombres premiers aléatoires inférieurs à n afin d'automatiser la génération des paramètres p et q du système RSA.

4) Fonction keys (int Pk [], int Sk [])

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5
6 int pgcd(int a, int b) {
7     int temp;
8     while (b != 0) {
9         temp = b;
10        b = a % b;
11        a = temp;
12    }
13    return a;
14 }
15
16 int premier(int n) {
17
18     if (n <= 1) return 0;
19     if (n == 2) return 1;
20     if (n % 2 == 0) return 0;
21     for (int i = 3; i * i <= n; i += 2) {
22         if (n % i == 0) return 0;
23     }
24     return 1;
25 }
26
27 void two_primes(int n, int prime[]){
28     int count = 0;
29     while(count < 2){
30         int x = rand() % n;
31         if (premier(x)){
32             prime[count] = x;
33             count++;
34         }
35     }
36 }
```

```
void key(int pk[], int sk[]){
    int p, q, n, phi, e, d;
    int prime[2];

    two_primes(50, prime);
    p = prime[0];
    q = prime[1];
    n = p * q;
    phi = (p-1) * (q-1);
    e = 3;

    while((gcd(e, phi) != 1) e++);
    for(d=1; d<phi; d++){
        if ((e * d) % phi == 1) break;
    }

    pk[0] = n;
    sk[0] = n;

    pk[1] = e;
    sk[1] = d;
}

int main() {
    int pk[2];
    int sk[2];

    srand(time(NULL));
    key(pk, sk);

    printf("== Cle publique ==\n");
    printf("n = %d\n", pk[0]);
    printf("e = %d\n", pk[1]);

    printf("\n== Cle secrete ==\n");
    printf("n = %d\n", sk[0]);
    printf("d = %d\n", sk[1]);

    return 0;
}
```

Nous avons développé une fonction permettant de générer automatiquement la clé publique et la clé privée afin de simuler le fonctionnement complet du protocole RSA.

5) Fonction `to_bin(int bin[], int n)`

```
#include <stdio.h>

void to_bin(int bin[], int n){

    int i = 0;

    while(n > 0){
        bin[i] = n % 2;
        n /= 2;
        i++;
    }

}

int main() {

    int n;
    int bin[32];
    int i = 0;

    printf("Entrez un entier : ");
    scanf("%d", &n);

    to_bin(bin, n);

    printf("Représentation binaire : ");

    while(n > 0){
        printf("%d", bin[i]);
        i++;
        n /= 2;
    }

    return 0;
}
```

Nous avons implémenté cette fonction pour convertir un entier en représentation binaire, ce qui facilite l'implémentation efficace de l'exponentiation modulaire.

6) Fonction `Expo_modulaire(int a, int b, int n)`

```
1 #include <stdio.h>
2
3
4 int expo_modulaire(int a, int b, int n){
5
6     int rest = 1;
7     a = a % n;
8
9     while(b > 0){
10
11         if(b % 2 == 1){
12             rest = (rest * a) % n;
13             b /= 2;
14             a = (a * a) % n;
15         }
16     }
17
18     return rest;
19 }
20
21 int main(){
22
23     int a, b, n;
24
25     printf("Entrez la valeur de a : ");
26     scanf("%d", &a);
27
28     printf("Entrez la valeur de b : ");
29     scanf("%d", &b);
30
31     printf("Entrez la valeur de n : ");
32     scanf("%d", &n);
33
34     int p = expo_modulaire(a, b, n);
35
36     printf("a^b est : %d", p);
37
38     return 0;
39 }
40
```

Nous avons programmé l'algorithme d'exponentiation modulaire rapide afin de calculer efficacement $a^b \bmod n$, opération essentielle pour le chiffrement et le déchiffrement RSA.

Conclusion générale

Ce TP nous a permis de comprendre que chaque calcul effectué dans RSA, du choix des nombres premiers au calcul de l'inverse modulaire, joue un rôle essentiel dans la sécurité et le bon fonctionnement du système.