Travail Pratique El-Gamal

Maxime Lovino Thomas Ibanez

16 janvier 2017

1 Introduction

Nous avons réalisé une implémentation de l'algorithme de signature d'El-Gamal en Matlab. Cette implémentation comprend la génération des clés publiques et privées, des fonctions d'arithmétique modulaire ainsi qu'une fonction de signature et de vérification de la signature.

2 Fonctions réalisées

2.1 Fonction modulo

```
function [ value ] = modulo( a,b )
%MODULO Return the result of a mod b

% We take the euclidean division of a and b
x = floor(a/b);
% We remove x times b from a to get the remainder
value = a - x*b;
end
```

Cette fonction retourne donc la valeur du modulo de a par rapport à b. Son implémentation est réalisée à l'aide de la division entière de a par b.

2.2 Fonction PGCD

```
function [ out ] = gcd( a,b )
%GCD Function to compute the gcd of two numbers
%   We use the euclidean algorithm for the gcd

if b == 0
    out = a;
else
    temp = b;
    b = modulo(a,b);
    a = temp;
    out = gcd(a,b);
end

end
```

Il s'agit ici de la version récursive de l'algorithme d'Euclide pour la recherche du PGCD de deux nombres. Nous remplaçons à chaque itération la valeur de b par le modulo de a par b, jusqu'à ce que b soit égal à 0. Dans ce cas, la valeur du PGCD est la valeur de a.

2.3 Fonction copremier

```
function [ out ] = coprime( a,b )
%COPRIME Function that returns 1 if a is coprime to b, 0 otherwise
    out = gcd(a,b) == 1;
end
```

Implémentation très simple d'une fonction booléenne pour vérifier si a et b sont copremiers, nous nous servons ici de la fonction PGCD écrite plus haut.

2.4 Fonction générateur

```
function [ gen ] = generator( p )
%GENERATOR Function that returns a generator of Z/pZ*
```

```
for i=2 :1 :p
%     Check array, zeros if value not obtained yet
    check = zeros(1,p-1);
    for j=0 :1 :p
%         We calculate the value, if it was already obtained, we stop this
%         loop, otherwise we write 1 in the array
        temp = modExp(i,j,p);
        if check(1,temp) == 1
            break;
        else
            check(1,temp) = 1;
        end
    end
%     if we obtained everything, it's a generator
    if check == ones(1,p-1)
        gen = i;
        return;
    end
end
    gen = -1;
end
```

Fonction qui retourne un générateur de $\mathbb{Z}/p\mathbb{Z}^*$. Pour ce faire nous testons toutes les valeurs potentielles de générateur et nous vérifions à l'aide d'un tableau que nous obtenons toutes les valeurs comprises dans $[1, p-1]$ en élevant les valeurs du candidat générateur aux puissances comprises dans $[0, p]$ et en prenant leur modulo par rapport à p. Dès que nous obtenons une valeur deux fois avant d'avoir rempli tout le tableau, nous pouvons passer au prochain candidat, car il ne s'agit pas d'un générateur. Nous renvoyons -1 si aucun générateur n'a pu être trouvé.

2.5 Fonction inverse modulaire

```
function [ inv ] = inverseMod( a, n )
%INVERSE_MOD Function that computes the inverse of a mod n (a^(-1) mod n)
%The algorithm used is the extended euclidean algorithm
    if(~coprime(a,n))
        inv = -1;
        return;
    end
    q = a;
    r = n;
    Q = [1, 0];
    R = [0, 1];
    qmodr = modulo(q, r);

    while(qmodr ~= 0)
        fqr = floor(q/r);
        T = Q-fqr*R;
        Q = R;
        R = T;
        q = r;
        r = qmodr;
        qmodr = modulo(q, r);
    end
    inv = modulo(T(1), n);
end
```

Implémentation de l'inverse modulaire de a par rapport à n basé sur la méthode vue en cours. Nous retournons -1 si l'inverse n'existe pas (si a et n sont copremiers)

2.6 Fonction exponentiation modulaire

```
function [ out ] = modExp( a,b,n )
%MODEXP Function that computes the modular exponentiation of a^b mod n

initA = a;
if b == 0
    out = 1;
    return;
end

for i=2 :b
    a = (initA*a);
    a = modulo(a,n);
end
out = a;
end
```

Implémentation de l'exponentiation modulaire

$$a^b \mod n$$

Ici, nous faisons d'abord un test, car si b vaut 0, il suffit de retourner 1 directement, puis ensuite, nous multiplions par a en prenant le modulo du résultat à chaque itération.

2.7 Fonctions pour nombres premiers

Ici nous regroupons plusieurs fonctions, parmi lesquelles un test de primalité par exemple, qui vont nous servir à générer un nombre premier aléatoire pour la génération de nos clés.

2.7.1 Test de miller

```
function [ pass ] = millerTest( a,n )
%MILLER_TEST Function that computes the miller test for n

    %find n-1 = 2^s * d
    for i=0 :floor(log2(n-1))+1
        if(modulo((n-1), (2.^i)) == 0)
            s = i;
            d = (n-1)/(2.^i);
        end
    end
    %apply miller test

    x = modExp(a, d, n);
    if((x == 1) || (x == n-1))
        pass = 0;
        return;
    end

    while(s > 1)
        x = modulo(x.^2, n);
        if(x == n-1)
            pass = 0;
            return;
        end
        s = s-1;
```

```
    end
    pass = 1;
end
```

Implémentation du test de Miller servant au test de primalité de Miller-Rabin énoncé ci-dessous.

2.7.2 Test de primalité

```
function [ prime ] = isPrime( n, k )
%ISPRIME Returns 1 if n is prime, 0 otherwise, tested over k iterations
%The implementation for this function is the Miller-Rabin test
    if(n == 2)
        prime = 1;
        return;
    end
    if(n <= 1 || modulo(n,2) == 0)
        prime = 0;
        return;
    end
    %we made sure that n is odd, which is a condition for the algorithm to
    %work
    for i=1 :k
        %a is randomly picked between [2, n-2]
        a = floor(rand()*(n-4)+2);
        %if the Miller test succeeds, n is not a prime
        if(millerTest(a, n))
            prime = 0;
            return;
        end
    end
    prime = 1;
end
```

Test de primalité, qui utilise l'algorithme de Miller-Rabin sur k itérations. Nous retournons un booléen spécifiant si n est premier ou pas.

2.7.3 Générateur de nombre premier aléatoire

```
function [ n ] = randomPrime( min, max )
%RANDOM_PRIME Function that returns a random prime in the interval
%[min:max]

x = 1;
while(~isPrime(x, 10))
    x = round(rand()*(max-min)+min);
end
n = x;
end
```

Fonction qui permet de générer un nombre premier aléatoire dans l'intervalle $[min, max]$ Nous tirons des nombres aléatoirement dans cet intervalle tant que nous ne tombons pas sur un nombre premier, le test de primalité sera celui écrit plus haut.

2.8 Fonction de génération des clés

```
function [ p, alpha,a,beta ] = generateKeys()
%GENERATE_KEYS Function that generates the keys, private and public
p = randomPrime(100,1000);
```

```
alpha = generator(p);
a =  round(rand()*(p-2)+1);
beta = modExp(alpha,a,p);
end
```

Cette fonction sert à générer toutes les valeurs composant les clés publiques et privées d'El-Gamal, la fonction retourne 4 valeurs : (p, α, a, β)

2.9 Fonction de signature

```
function [ gamma,delta ] = signature( x, alpha, p, a)
%SIGNATURE Function to sign using El-Gamal

k = round(rand()*(p-2)+1);

while(~coprime(k,p-1))
    k = round(rand()*(p-2)+1);
end

gamma = modExp(alpha,k,p);

delta = modulo((x-a.*gamma)*inverseMod(k,p-1),p-1);

end
```

Cette fonction va signer le message x en utilisant à l'aide des clés générées plus haut (plus précisément la clé privée a, ainsi que α et p). Elle va nous renvoyer les valeurs de γ et δ

2.10 Fonction de vérification de la signature

```
function [ out ] = signatureCheck( delta, gamma, beta, alpha,p,x )
%SIGNATURE_CHECK Returns 1 if the signature is valid for the message x, 0
%otherwise
out = modulo(modExp(beta,gamma,p)*modExp(gamma,delta,p),p) == modExp(alpha,x,p);
end
```

Fonction qui va vérifier la signature et va nous retourner une valeur booléenne pour attester de sa validité, elle prend en paramètre le message x ainsi que la clé publique et la signature générée.

3 Exemple d'exécution

```
>> [p,alpha,a,beta] = generateKeys

p =

   857


alpha =

     3


a =

   711
```

```
beta =

   431

>> message = 42

message =

    42

>> [gamma,delta] = signature(message,alpha,p,a)

gamma =

   504


delta =

   474

>> signatureCheck(delta,gamma,beta,alpha,p,message)

ans =

     1
```

4 Conclusion

En conclusion, on peut dire que ce TP nous a aidé à mieux comprendre la théorie vue en cours. Nous avons également eu l'opportunité d'implémenter un algorithme de test de primalité, ce qui pourra nous servir par la suite dans d'autres travaux.