Travail Pratique El-Gamal

Maxime Lovino Thomas Ibanez

15 janvier 2017

1 Introduction

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

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
```

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
```

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
```

2.4 Fonction générateur

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

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
```

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)
    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
```

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
```

2.7 Fonctions pour nombres premiers

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
```

2.7.2 Test de primalité

```
function [ prime ] = isPrime( n, k )
%ISPRIME Returns 1 if n is prime, 0 otherwise, tested over k iterations
    if(n == 2)
        prime = 1;
        return;
    end
    if(n <= 1 || modulo(n,2) == 0)
        prime = 0;
        return;
    end
    for i=1 :k
        a = floor(rand()*(n-4)+2);
        if(millerTest(a, n))
            prime = 0;
            return;
        end
    end
    prime = 1;
end
```

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
```

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
```

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
```

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
```

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

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.