Teeny-Tiny Implementation of Multivariate Polynomial Public Key / Digital Signature (MPPK/DS)

Work in Progress

Author: Michel Barbeau

Version: January 4, 2022

```
clear;
```

Reference:

Randy Kuang, Maria Perepechaenko, and Michel Barbeau, A New Quantum Safe Multivariate Polynomial Public Key Digital Signatures Algorithm, December 2021.

Security Parameters

```
% Finite Field index
p = 257;
% Euler's totient of p
tp = p - 1;
% number of noise variables
m = 2;
% degree of base polynomial
n = 2;
% degree of multiplier polynomials
lambda = 1; % linear
% upper limits
ell = [ 1 1 ];
```

Soundness Assessment

```
j=0;
for i=1:100
    verdict = MPPKDS(m,n,lambda,ell,p);
    if strcmp(verdict,'INVALID')
        fprintf('%d ', i); j = j + 1;
    end
end
fprintf('\n');
fprintf('j=%d\n',j);
```

```
function verdict = MPPKDS(m,n,lambda,ell,p)
```

Random message μ

```
    mu = randi([0 p-1],1);
%     disp(mu)
    [s,v] = K(m,n,lambda,ell,p);
```

```
%     disp(s)
%     disp(v)
    [mu,t] = S(s,mu,m,n,lambda,ell,p);
    disp(t); % [A B C D E]
    % A = t(1); B = t(2); C = t(3); D = t(4); E = t(5);
    % if (A*B*C*D*E)==0 | ((A-1)*(B-1)*(C-1)*(D-1)*(E-1))==0
    %     fprintf('A, B, C, D and E must not be equal to 0 or 1');
    %     return;
    % else
    [mu,verdict] = V(v,mu,t,m,n,lambda,ell,p);
    disp(verdict);
    % end
end
```

Key Generation Algorithm

```
function [s,v] = K(m,n,lambda,ell,p)
    % m = number of noise variables
    % n = degree of base polynomial
    % lambda = degree of univariate polynomials
    % ell = upper limits, in base polynomial
    % p = finite field index
    tp = p - 1; % Euler's totient of p
```

1. Base Polynomial $\beta(x_0, x_1, \ldots, x_m)$

```
    % randomly generate coefficients for beta()
    c = randi([0 tp-1], n+1, (ell(1)+1)*(ell(2)+1) );
    % c = zeros(n+1, (ell(1)+1)*(ell(2)+1) );
```

2. Univariate ploynomials $f(x_0)$ **and** $h(x_0)$

```
    % randomly generate coefficients of f()
    f = randi([0 tp-1], 1, lambda+1);
    % randomly generate coefficients of h()
    h = randi([0 tp-1], 1, lambda+1);
```

3. Product polynomials $\phi(x_0, x_1 \ldots, x_m)$ **and** $\psi(x_0, x_1 \ldots, x_m)$

```
    % init \phi and \psi to zeros
    phi = zeros(n+lambda+1, (ell(1)+1)*(ell(2)+1));
    psi = zeros(n+lambda+1, (ell(1)+1)*(ell(2)+1));
    for i=0:n
        for j=0:lambda
            phi(i+j+1,:) = phi(i+j+1,:) + c(i+1,:).*f(j+1);
            psi(i+j+1,:) = psi(i+j+1,:) + c(i+1,:).*h(j+1);
        end
    end
    phi = mod(phi,tp);
    psi = mod(psi,tp);
```

4. Polynomials $E_\phi(x_0)$ and $E_\psi(x_0)$

```
    % randomly generate coefficients
    Ephi = randi([0 tp-1], 1, n+lambda-1);
    % randomly generate coefficients
    Epsi = randi([0 tp-1], 1, n+lambda-1);
```

5. R_0, and R_n

```
    R0 = randi([1 (tp-2)/2],1).*2;
    Rn = randi([1 (tp-2)/2],1).*2;
```

6. Noise functions $N_0(x_1,\ldots,x_m)$ and $N_n(x_0,x_1,\ldots,x_m)$

```
    N0 = mod(R0 * c(1,:),tp);
    Nn = mod(Rn * c(n+1,:),tp);
```

7. $\Phi(x_0,x_1,\ldots,x_m)$ and $\Psi(x_0,x_1,\ldots,x_m)$

```
    Phi = phi(2:n+lambda,:);
    Psi = psi(2:n+lambda,:);
```

8. $P(x_0,x_1,\ldots,x_m)$ and $Q(x_0,x_1,\ldots,x_m)$

```
    P = zeros(n+lambda-1,(ell(1)+1)*(ell(2)+1));
    Q = zeros(n+lambda-1,(ell(1)+1)*(ell(2)+1));
    for i=1:(n+lambda-1)
        P(i,:) = R0 * ( [ Phi(i,1)-Ephi(i) Phi(i,2:end) ]);
        Q(i,:) = Rn * ( [ Psi(i,1)-Epsi(i) Psi(i,2:end) ]);
    end
    P = mod(P,tp);
    Q = mod(Q,tp);
```

Private-key and public-key pair (s,v)

```
    s = { f h R0 Rn Ephi Epsi };
    v = { P Q N0 Nn };
end
```

Signing Algorithm

```
function [mu,t] = S(s,mu,m,n,lambda,ell,p)
    % t = digital signature
    % mu = message
    % s = private key
    f = cell2mat(s(1)); h = cell2mat(s(2)); R0 = cell2mat(s(3));
    Rn = cell2mat(s(4)); Ephi = cell2mat(s(5)); Epsi = cell2mat(s(6));
    % m = number of noise variables
    % n = degree of base polynomial
    % lambda = degree of univariate polynomials
```

```
    % ell = upper limits, in base polynomial
    % p = finite field index
    tp = p - 1; % Euler's totient of p
```

Random base

```
    g = randi([2 tp-1],1);
```

Evaluate $f(x_0)$ on μ

```
    fm = polyval(flip(f),mu);
    a = mod(R0*fm,tp);
    A = powermod(g,a,p);
```

Evaluate $h(x_0)$ on μ

```
    hm = polyval(flip(h),mu);
    b = mod(Rn*hm,tp);
    B = powermod(g,b,p);
    c = mod(Rn * ( hm*f(1) - fm*h(1) ),tp);
    C = powermod(g,c,p);
    d = mod(R0 * ( hm*f(lambda+1) - fm*h(lambda+1) ),tp);
    D = powermod(g,d,p);
```

Evaluate $E_\phi(x_0)$ on μ

```
    Ephim = polyval([flip(Ephi) 0],mu);
    Ephim = mod(Ephim,tp);
```

Evaluate $E_\psi(x_0)$ on μ

```
    Epsim = polyval([flip(Epsi) 0],mu);
    Epsim = mod(Epsim,tp);
    e = mod(R0*Rn*(hm*Ephim - fm*Epsim),tp);
    E = powermod(g,e,p);
    % digital signature
    t = [ A B C D E ];
end
```

Signature Verifying Algorithm

```
function [mu,verdict] = V(v,mu,t,m,n,lambda,ell,p)
    % v = public key
    P = cell2mat(v(1)); Q = cell2mat(v(2)); N0 = cell2mat(v(3)); Nn = cell2mat(v(4));
    % mu = message
    % t = digital signature
    A = t(1); B = t(2); C = t(3); D = t(4); E = t(5);
    % m = number of noise variables
    % n = degree of base polynomial
    % lambda = degree of univariate polynomials
    % ell = upper limits, in base polynomial
    % p = finite field index
```

```
    tp = p - 1; % Euler's totient of p
```

Noise variables $r_1, \ldots, r_m$

```
    r = randi([1 tp-1],1,m);
    % r = [2 2]
```

Evaluate $\bar{P} = P(m, r_1, \ldots, r_m)$, $\bar{Q} = Q(m, r_1, \ldots, r_m)$, $\bar{N}_0 = N_0(r_1, \ldots, r_m)$ and $\bar{N}_n = N_n(x_0, r_1, \ldots, r_m)$

```
    barP = 0; barQ = 0;
    for i=1:(n+lambda-1)
        for j1=0:ell(1)
            for j2=0:ell(2)
                barP = mod(barP + P(i,(j1*2)+j2+1)*powermod(r(1),j1,tp)*powermod(r(2),j
                barQ = mod(barQ + Q(i,(j1*2)+j2+1)*powermod(r(1),j1,tp)*powermod(r(2),j
            end
        end
    end
    barP = mod(barP,tp);
    barQ = mod(barQ,tp);
    barN0 = 0; barNn = 0;
    for j1=0:ell(1)
        for j2=0:ell(2)
            barN0 = mod(barN0 + N0((j1*2)+j2+1)*powermod(r(1),j1,tp)*powermod(r(2),j2,t
            barNn = mod(barNn + Nn((j1*2)+j2+1)*powermod(r(1),j1,tp)*powermod(r(2),j2,t
        end
    end
    barN0 = mod(barN0,tp);
    barNn = mod(barNn*powermod(mu,(n+lambda),tp),tp);
    % Verification
    left = powermod(A,barQ,p);
    right = mod(powermod(B,barP,p)*powermod(C,barN0,p)*powermod(D,barNn,p)*E,p);
    if left==right
        verdict = 'VALID';
    else
        verdict = 'INVALID';
    end
end
```