Companion software for "Volker Ziemann, *Physics and Finance, Springer, 2021*" (https://link.springer.com/book/10.1007/978-3-030-63643-2)

Elliptic-curve signature with ECDSA (Section 12.7)

This example illustrates the verification of digital signatures based on elliptic-curve cryptography (ECDSA) and is based on Exercise 8 in Chapter 12 and its solution from Chapter 13. In the exercise Alice sends Bob a message (either 'YES' or 'NO') and creates a ECDSA signature (r,s) which she also sends to Bob, who uses Equation 12.35 to verify the authenticity of the message received.

Alice and Bob first agree on the elliptic curve with parameters a and b over an integer field based on a prime p and a generator point G, which was previously derived in `EllipticCurveArithmetic.mlx` where we also determined that the order of the group, generated by G, is n. All these parameters are publicly known.

```
p=113;
a=0;
b=7;
G=[15,52];
n=19;
```

Alice signs a message

Alice calculates the signature of her message m by calculating a hash value h (we use a very simple one based on xor).

```
m='YES';   % or m='NO';       % message
nm=double(m);                 % numeric form
h=bitxor(nm(3),bitxor(nm(2),nm(1)));
```

Then she comes up with her secret key d and calculates her private point P by adding G, the generator point, d times.

```
d=15;                         % Alice's private key
P=G; for j=2:d, P=ECadd_p(P,G,a,b,p); end
```

The first part of the signature is derived from some random number k, summing G a number of k times to obtain a point R, whose x-coordinate is first part of the signature.

```
k=4;                          % random number
R=G; for j=2:k, R=ECadd_p(R,G,a,b,p); end
r=R(1)                        % 1. part of signature
```

```
r = 66
```

The second part of the signature s is calculated from the hash as $s=(h+rd)/k \pmod n$. The inverse of k is calculated with Equation 12.34.

```
kinv=powermod(k,n-2,n);       % 1/k
s=powermod((h+r*d)*kinv,1,n)  % 2. part of signature
```

```
s = 6
```

Alice now sends the message `m` and the signature (r, s) to Bob.

Bob verifies the signature

Now Bob must verify the message he received in plaintext in order to be sure that Eve, who wants to mess up the relation between Alice and Bob, did not fake the message. He therefore first calculates the hash h of the received message `m` using the same hash function as Alice.

```
m='YES';   % or m='NO';       % received message
nm=double(m);                 % numeric form
h=bitxor(nm(3),bitxor(nm(2),nm(1)));  % hash value
```

And then he uses Equation 12.35 to calculate Q, whose x-coordinate should agree with r, the x-coordinate of R that Alice had derived and used as part of her signature.

```
sinv=powermod(s,n-2,n);       % 1/s
hs=powermod(h*sinv,1,n);      % h/s
rs=powermod(r*sinv,1,n);      % r/s
Q1=G; for j=2:hs, Q1=ECadd_p(Q1,G,a,b,p); end  % (h/s)*G
Q2=P; for j=2:rs, Q2=ECadd_p(Q2,P,a,b,p); end  % (r/s)*P
Q=ECadd_p(Q1,Q2,a,b,p)        % (h/s)*G+(r/s)*P
```

```
Q = 1×2
    66   101
```

And here `Q(1)` should be the same as the first part of the signature `r`. Note that the purpose of the signature is an elaborate scheme to transmit Alice's secret information across an unsafe communication channel, without actually divulging the secret, yet giving anyone enough infromation to verify the authenticity of the message, because only Alice could have prepared a signature that is consistent with the received message.

Appendix

The function `ECadd_p()` receives two points `Pa` and `Pb` on an elliptic curve, which is specified by parameters `a` and `b`, as well as the base `p`. It returns a new point on the same curve `Pc` that is calculated from Equation 12.32 and 12.33. The handling of special cases is discussed in Section 12.7.

```
function Pc=ECadd_p(Pa,Pb,a,b,p)
if Pa(2) == Inf, Pc=Pb; return; end
if Pb(2) == Inf, Pc=Pa; return; end
if Pa(1) == Pb(1) & Pa(2) ~= Pb(2), Pc=[0,Inf]; return; end
if Pa(1) == Pb(1)
  if Pa(2) == Pb(2) & Pa(2) == 0
    Pc=[0,Inf];
    return
  else
    denominv=powermod(2*Pa(2),p-2,p);  % was 1
    s=powermod((3*Pa(1)*Pa(1)+a)*denominv,1,p);
  end
else
```

```
  denominv=powermod(Pb(1)-Pa(1),p-2,p);
  s=powermod((Pb(2)-Pa(2))*denominv,1,p);
end
Pc(1)=powermod(s*s-Pa(1)-Pb(1),1,p);
Pc(2)=powermod(-s*(Pc(1)-Pa(1))-Pa(2),1,p);
end
```