

***SCHOOL OF ENGINEERING &TECHNOLOGY***

***BML MUNJAL UNIVERSITY***

***April 2020***

***Code Documentation***

***Topic: Elliptical Curve Cryptography***

**Submitted By:**

Rahul Gupta(1700161C202)

Ayush Govil(1700120C202)

**Branch:** CSC

**Coding Language: Python**

File Name: eclib.py

Libraries used: collections

Functions:

* inv(n, q)
* egcd(a, b)
* inv2(n, q)
* sqrt(n, q)
* jacobi(a,q)
* jacobi2(a,q)
* is\_valid(self, p)
* at(self, x)
* neg(self, p)
* add(self,p1,p2)
* mul(self,p,n)
* order(self,g)
* gen(self,priv)
* secret(self, priv, pub)
* validate(self, hashval, priv, r)

Explanation:

* **def \_\_init\_\_(self, ec, g):** Constructor to initialize ec which represents elliptic curve in a Finite Field.
* **self.ec=ec:** ec is the instance of the EC class.
* **self.g=g:** g is the generator point on the ec. The point multiplication is done from this point.
* **self.n= ec.order(g):** n is the prime order of G. The value of private key is from 1 to n-1
* **assert 0 < priv and priv < self.n:** It checks whether the private key value is within the range from 1 to n-1 where n is the prime order of the generator point
* **assert self.ec.is\_valid(pub):** This function checks whether the public key which is exchanged between the sender and receiver is valid or not.
* **assert self.ec.mul (pub, self.n) == self.ec.zero:** This function generates the final key which is given as: N.[q] where N is [p].g which was calculated on the other side and was sent by the sender and g is the private key.

Point multiplication in ECC is commutative therefore: [p].n.[q] = [q].n.[p] therefore on the other side the public key will be [q].n and the private key will be [p] so the same key is generated.

File Name: ecdsa.py

Libraries used: collections

hashlib

Functions:

* inv(n, q)
* sqrt(n, q)
* is\_valid(self,p)
* at(self,x)
* neg(self,p)
* add(self,p1,p2)
* mul(self,p,n)
* order(self,g)
* gen(self,priv)
* sign(self, hashval, priv,r)
* validate(self, hashval, priv, r)

Explanation:

* **g, \_ = ec.at(7):** Producing a generator point
* **hashval = int("0x" + hashlib.md5(msg).hexdigest(), 16):** Encrypting message at sender’s end.
* **sig = dsa.sign(hashval, priv, r):** Used to generate signature
* **hashval\_rec = int("0x" + hashlib.md5(msg\_rec).hexdigest(), 16):** Decrypting message at receivers end.
* **if dsa.validate(hashval\_rec, sig, pub) == True:** Checking whether digital signature matches or not.

**NOTE:** Every function for both files contains comments in code file where we have mentioned what every variable represents and explained every function’s use.