## Part B

**Sign and validate transactions by DSA (i.e. DSS, the signing equation defined in NIST FIPS 186-4): implement a DSA module which enables the user can sign the transactions and a miner can verify the signed transaction. (Note that since we set all inputs to zero, so $m_i = Tx_i$, $i$ = 1, 2, 3.)**

1) **User 1: sign his transaction: $Sig_{sk_1}(m_1) = (r_1, s_1)$ i.e, compute the signature over $m_1, k_1, r_1 = \left(g^{k_1} mod(p)\right) mod(q), k_1{}^{-1} mod(q)$ and $s = k_1{}^{-1}\left(h(m_1)\right) + xrmod(q)$**

In accordance with the NIST FIPS 186-4 specification, the signature generation function was implemented using the following function provided in section 4.6 DSA Signature Generation:

```
# DSA signature function, p, q, g, k, sk are integers, Message are hex strings of even length.
def Sign( p, q, g, k, sk, Message ):
    k_inv = inverse_mod(k, q)
    if k_inv is not None:
        r = pow(g, k, p) % q

        N = len(bin(q)[2:])

        h_m = sha3_224_hex(Message)
        binary_h_m = str(bin(int(h_m, 16)))[2:].zfill(len(h_m[2:] * 4))
        outlen = len(binary_h_m[2:])
        z = binary_h_m[:min(N, outlen)]

        z_10 = int(z, 2)
        s = (k_inv * (z_10 + (sk * r))) % q

        if r != 0 and s != 0:
            return r,s

        raise Exception("either r or s were equal to 0 - generate a new value for k")
    raise Exception('k-1 was not found')
```

The set of input parameters *p*, *q*, *g*, $k_1$, $sk_1$, and $m_1$, and output parameters $r_1$ and $s_1$ can be found in the Appendix.

2) **A miner: verify $Sig_{sk_1}(m_1)$, i.e., compute the values of $u$, $v$, $w$, and verify whether $v$ = $r$.**

In accordance with the NIST FIPS 186-4 specification, the signature verification function was implemented using the following function provided in section 4.7 DSA Signature Verification and Validation:

```
# DSA verification function,  p, q, g, k, pk are integers, Message are hex strings of even length.
def Verify( p, q, g, pk, Message, r, s ):
    M_prime = Message
    r_prime = r
    s_prime = s
    y = pk
    N = len(bin(q)[2:])

    if not (0 < r_prime < q) or not (0 < s_prime < q):
        return False

    w = inverse_mod(s_prime, q)
    # print('w: ' + str(w))
    if w is not None:
        h_m = sha3_224_hex(M_prime)
        binary_h_m = str(bin(int(h_m, 16)))[2:].zfill(len(h_m[2:] * 4))
        outlen = len(binary_h_m[2:])
        z = binary_h_m[:min(N, outlen)]
        z_10 = int(z, 2)

        u1 = (z_10 * w) % q
        # print('u1: ' + str(u1))
        u2 = (r_prime * w) % q
        # print('u2: ' + str(u2))
        v1 = pow(g, u1, p)
        v2 = pow(y, u2, p)
        v = ((v1 * v2) % p) % q
        # print('v: ' + str(v))

        return True if v == r_prime else False
```

The set of input parameters $p$, $q$, $g$, $pk1$, $m_1$, $r$, and $s$ can be found in the Appendix. The output produced by the set of input parameters was True. The intermediate variables $w$, $u_1$, $u_2$, and $v$ can be found in the Appendix.

3) **User 2: sign his transaction $m_2 = Tx_2$ using the key pair ($sk_2$, $pk_2$), i.e. executing the same steps as user 1.**

Using the function from part 1) The set of input parameters $p$, $q$, $g$, $k_2$, $sk_2$, and $m_2$, and output parameters $r_2$ and $s_2$ can be found in the Appendix.