```
# Symmetric key ratchet
# Take the current key and some value to generate the next key using a one-way
# key-derivation function
# Non reversible to provide forward security

# initialise ratchet state in server
def sym_key_ratchet(key, value=b ‘’, server)
        output = HKDF(algorithm=hashes.SHA256(), length=80, salt=b ‘’,
            info=b ‘’, backend=default_backend()).derive(key + value)
        # change the state of ratchet
        server.state = output[:32]
        # generate new key outkey and IV
        outkey, iv = output[32:64], output[64:]
      return {
        ‘outkey’ = outkey
        ‘iv’ = iv
        ‘state’ = state
      }

def init_ratchet(shared_key):
        root_ratchet = sym_key_ratchet(shared_key)[‘state’]
        send_ratchet = sym_key_ratchet(shared_key)[‘outkey’]
        recv_ratchet = sym_key_ratchet(shared_key)[‘outkey’]
        return root_ratchet, send_ratchet, recv_ratchet
# In alice and bob, initialise variable

# In alice and bob,  initialise public key dhr, dh_recv, dh_send

def dh_ratchet(ratchet_key_sender, public_key_recipient, param):
        # Use public key of interlocutor to generate new key
        dh_send = Keygen.dh(param, ratchet_key_sender, public_key_recipient)
        # Sender applies a symmetric-key ratchet step to her sending chain key
        # New chain key is stored, message key and old chain key can be deleted
        sk_send = sym_key_ratchet(dh_send)[‘outkey’]
        # update sending chain with new key
        send_ratchet = sym_key_ratchet(sk_send)
        # Sender receives a response from recipient with a new ratchet public key
        if dh_ratchet != 0
        # Sender applies a DH ratchet step to derive new receiving chain keys
        dh_recv = Keygen.dh(p, ratchet_key_sender, public_key_recipient)
        # Sender applies a symmetric-key ratchet step to the receiving chain to get
        # the message key for the received message
        sk_recv = sym_key_ratchet(dh_recv)[‘outkey’]
        recv_ratchet = sym_key_ratchet(sk_recv)
        return send_ratchet, recv_ratchet

def send(sender, recipient, message):
   key, iv, state =sender.send_ratchet()
```

```
    # AES CBC Cipher Block Chaining
    cipher = AES.new(key, AES.MODE_CBC)
    cipher_text = cipher.encrypt(msg)
    # send current DH public key
   recipient.recv(recipient, cipher,sender.dh_ratchet_key)
    print('Encrypted message :', cipher_text)
    return key, iv

def recv(recipient, cipher,public_key):
        decrypt_cipher = AES.new(key, AES.MODE_CBC, iv)
        plain_text = decrypt_ciper.decrypt(cipher_text)
        recipient.dh_ratchet(key)
        key, iv = recipient.recv_ratchet()
        print('Decrypted message:', plain_text)
        return key, iv


######## Communication

# initialise alice, bob, server

# perform x3dh

# initialiser
alice.root_ratchet, alice.send_ratchet, alice_recv_ratchet = init_ratchet(alice.Eka)

bob.root_ratchet, bob.send_ratchet, bob_recv_ratchet = init_ratchet(bob.Ekb)

# alice perform dh ratchet
dh_ratchet(alice.dh_ratchet_key, bob.

send(alice, bob, 'hello')
```