# Security 1 - Mandatory handin 2

**1. Design a protocol that allows Alice an Bob to throw a virtual 6 sided dice over the insecure network even though they do not trust each other without allowing for an adversary to see that they are playing dice.**

Alice and Bob are playing a game of dice even though they don’t trust each other. The game I have setup has the following rules:

- Alice has to try and guess what the die roll will be

Is this fair to Alice when she only has a ⅙ chance of being right? Depends, what if when Alice lose she has to pay Bob 5$, but if she guess correctly he will pay her 10.000$. Now this is not fair to Bob. All this to say, I don’t care how they make this game fair. I just set up the rule that Alice has to try and guess the correct number.

## The program works as so:

1. Bob works as the server, forever listening if Alice wants to play dice
   (Bob.py)

```python
#====================== START SERVER ======================
def start():
    server.listen()
    print(f"[SERVER LISTINING] on {SERVER}")
    while True:
        conn, addr = server.accept() #Will wait for at new connection
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()
```

2. Alice tells Bob she wants to roll a dice
3. Bob decides on and sends Alice the genarators g and p
   (Bob.py)

```python
#====================== AGREE UPON G AND P ======================
def setup(conn, addr):
    msg = server_recive(conn, addr)
    if msg == "[ROLL THE DICE!]":
        g = generate_prime(100)
        p = generate_prime(50)
        server_send(conn, addr, str(g))
        server_send(conn, addr, str(p))
        print("[PARAMETERS SETUP]")
        return g, p
```

4. Alice makes a guess on what she thinks the dice will land on
5. Alice uses these and another random int to hash her guess and sends this hashed guess to Bob
   (Alice.py)

```python
# ======================= COMMIT TO GUESS =======================
def commit(g, p):
    a = random.randint(1, 6)
    print(f"[ALICE GUESS] = {a}")
    h = g ** a % p
    r = generate_prime(10)
    c = (g ** a) * (h ** r)
    # print(f"[ALICE COMMITED] = {c}")
    send(str(c))
    return r, a
```

6. Both Bob and Alice commits (in much the same way as Alice's guess) a number that will be used to generate the die roll
7. Alice and Bob both send their numbers so that they can open the random number and calculate the roll
   (Alice.py)

```python
# ======================= CALC DICE ROLL ==========================
def CalcRoll(g, p):
    aR = random.randint(1, 20)
    h = g ** aR % p
    r = generate_prime(5)
    aC = (g ** aR) * (h ** r)  # Alice contribution to the dice roll
    send(str(aC))
    bC = int(receive())
    send(str(aR))
    send(str(r))
    bR = int(receive())
    br = int(receive())
    if opencommit(g, p, br, bR, bC):
        send("ACK")
        isAck = receive()
        if isAck == "ACK":
            roll = ((aR + bR) % 6) + 1
            print(f"[ROLL WAS]: {roll}")
            return roll
    else:
        send("NOT_ACK")
        isAck = receive()
        return 0
```

8. Alice sends the unhashed guess and the random int to Bob
9. Bob does the same hashing to see if he gets the same number as Alice's hashed commit
   (Bob.py)

```python
#======================= OPEN COMMIT ======================
def open(g, p, r, a, c):
    hh = (g**a) % p
    cc = (g**a)*(hh**r)
    return (cc == c)
```

10. If it matches Bob Accepts the guess and tells that to Alice
    (Bob.py)

```python
            r = int(server_recive(conn, addr))
            a = int(server_recive(conn, addr))
            print(f"[RECIVED] randint {r}, Alice guess {a}")
            if open(g, p, r, a, c):
                print("[COMMIT ACCEPTED]")
                server_send(conn, addr, "ACCEPTED")
                result(conn, addr, a, roll)
            else:
                print("[COMMIT NOT ACCEPTED]")
                server_send(conn, addr, "NOT ACCEPTED")
```

11. If Alice guessed correct she gets a message saying she got it

```python
#==================== SEND RESULT ======================
def result(conn, addr, a, d):
    if a == d:
        print("[ALICE GUESSED CORRECT]")
        server_send(conn, addr, "[ALICE GUESSED CORRECT]")
    else:
        print("[ALICE DID NOT GUESS CORRECT]")
        server_send(conn, addr, "[ALICE DID NOT GUESS CORRECT]")
```

**2. Explain why your protocol is secure using concepts from the lectures.**

**Secure Communication:**

Since libraries are allowed I use the Python libraries Sockets and SSL. I use sockets to setup a connection between Alice and Bob on localhost. The setup of sockets I use works on TCP. To make this secure I added the SSL library on top to have TLS. Using the SSL library I wrap the sockets so that when establishing a connection they follow the handshake protocol. All the details of establishing the TLS version , the compression method, the cipher suit, how to validate the identity of server and the client works and the generation of session keys is taken care of by this library.

This library also ensures confidentiality through cryptography of all messgaes. Integrity through message digests. And authenticity through digital certificates where it adds MAC to the messages before encrypting them. To make it easy for myself I use self signed certificates made using OpenSSL.

(Read more about the SSL library here:
https://docs.python.org/3/library/ssl.html#certificate-handling )

**Trust in the game:**

Before they roll a die Alice has to guess a number she thinks the die will land on. To ensure that either of them don't cheat Alice sends a commitment to Bob that he kan check to see that Alice is not lying about her guess after the die is rolled. She hides it so that Bob can not try and influence the roll since he doesn't know her guess.

This is done using Pedersen Commitment:

1. **Setup**: Agree upon generators
   (Bob.py)

```python
#================== AGREE UPON G AND P ====================
def setup(conn, addr):
    msg = server_recive(conn, addr)
    if msg == "[ROLL THE DICE!]":
        g = generate_prime(100)
        p = generate_prime(50)
        server_send(conn, addr, str(g))
        server_send(conn, addr, str(p))
        print("[PARAMETERS SETUP]")
        return g, p
```

2. **Commit**: Alice will choose a number between 1 and 6 as her guess. Then find a random number and then hash those to get the commitment she will send to Bob:
   $c = \text{generator\_g}^{(\text{message})} * \text{generator\_h}^{(\text{random\_number})}$

```python
# ================== COMMIT TO GUESS ====================
def commit(g, p):
    a = random.randint(1, 6)
    print(f"[ALICE GUESS] = {a}")
    h = g ** a % p
    r = generate_prime(10)
    c = (g ** a) * (h ** r)
    # print(f"[ALICE COMMITED] = {c}")
    send(str(c))
    return r, a
```

3. **Reveal**: After the dice is rolled Alice will reveal her guess by sending Bob her guess and random number. Bob will then do the same computation:
   $c = \text{generator\_g}^{(\text{message})} * \text{generator\_h}^{(\text{random\_number})}$
   He will then compare the commitment he got from her and the number he himself calculated and see if they are the same. I Alice tried to cheat they will not be.
   (Bob.py)

```python
#================== OPEN COMMIT ====================
def open(g, p, r, a, c):
    hh = (g**a) % p
    cc = (g**a)*(hh**r)
    return (cc == c)
```

Before she reveals her guess they roll the die. To ensure that no one cheats during this they also use commits here in much the same way. Both Alice and Bob commits to a number that will influence the final roll that they send hashed to eachother:
(Bob.py)

```python
#==================== CALC DICE ROLL ======================
def CalcRoll(g, p, conn, addr):
    bR = random.randint(1, 20)
    h = g**bR % p
    r = generate_prime(5)
    bC = (g**bR)*(h**r) #Bob contribution to the dice roll
    aC = int(server_recive(conn, addr))  #Alice commited contribution to the dice roll
    server_send(conn, addr, str(bC))
```

After they both committed they send the needed numbers to eachother so they can open eacothers commits:

```python
    aR = int(server_recive(conn, addr))
    ar = int(server_recive(conn, addr))
    server_send(conn, addr, str(bR))
    server_send(conn, addr, str(r))
    if open(g, p, ar, aR, aC):
        isAck = server_recive(conn, addr)
        server_send(conn, addr, "ACK")
        if isAck == "ACK":
```

Then, if they both acknowledge that no one cheated they calculate the die roll by:
((Alice_number + Bob_number) % 6) + 1

Like this they can ensure that Alice didn't cheat with her guess and that neither of them could screw the die to their advantage.

**3. Implement your virtual dice protocol in a programming language of your choosing.**

Bob works as the server and can stay open. Alice works as the client and has to be ran every time she wants to roll a die.

**To run the program:**
In one terminal write: python .\Bob.py
In another write: python .\Alice.py

Example of rolling the die a couple of times:

```
[CLIENT CONNECTED]: ('127.0.0.1', 55480)
[PARAMETERS SETUP]
[RECIVED COMMIT]
[ROLL WAS]: 1
[RECIVED] randint 829, Alice guess 3
[COMMIT ACCEPTED]
[ALICE DID NOT GUESS CORRECT]

[CLIENT CONNECTED]: ('127.0.0.1', 55481)
[PARAMETERS SETUP]
[RECIVED COMMIT]
[ROLL WAS]: 1
[RECIVED] randint 1013, Alice guess 2
[COMMIT ACCEPTED]
[ALICE DID NOT GUESS CORRECT]

[CLIENT CONNECTED]: ('127.0.0.1', 55482)
[PARAMETERS SETUP]
[RECIVED COMMIT]
[ROLL WAS]: 4
[RECIVED] randint 809, Alice guess 5
[COMMIT ACCEPTED]
[ALICE DID NOT GUESS CORRECT]

[CLIENT CONNECTED]: ('127.0.0.1', 55483)
[PARAMETERS SETUP]
[RECIVED COMMIT]
[ROLL WAS]: 4
[RECIVED] randint 563, Alice guess 1
[COMMIT ACCEPTED]
[ALICE DID NOT GUESS CORRECT]

[CLIENT CONNECTED]: ('127.0.0.1', 55489)
[PARAMETERS SETUP]
[RECIVED COMMIT]
[ROLL WAS]: 5
[RECIVED] randint 797, Alice guess 5
[COMMIT ACCEPTED]
[ALICE GUESSED CORRECT]
```

```
eth  master ≡ +1 ~1 -0 !                                   [19:53]
> python .\Alice.py
[SETUP COMPLETE]
[ALICE GUESS] = 3
[ROLL WAS]: 1
[COMMIT ACCEPTED]
[ALICE DID NOT GUESS CORRECT]
death@THATONE2  ~\Documents\Skole\ITU\5. Semester\Security 1\Assignments\Mandatory 2\Python\MH2_d
eth  master ≡ +1 ~1 -0 !                                   [19:54]
> python .\Alice.py
[SETUP COMPLETE]
[ALICE GUESS] = 2
[ROLL WAS]: 1
[COMMIT ACCEPTED]
[ALICE DID NOT GUESS CORRECT]
death@THATONE2  ~\Documents\Skole\ITU\5. Semester\Security 1\Assignments\Mandatory 2\Python\MH2_d
eth  master ≡ +1 ~1 -0 !                                   [19:54]
> python .\Alice.py
[SETUP COMPLETE]
[ALICE GUESS] = 5
[ROLL WAS]: 4
[COMMIT ACCEPTED]
[ALICE DID NOT GUESS CORRECT]
death@THATONE2  ~\Documents\Skole\ITU\5. Semester\Security 1\Assignments\Mandatory 2\Python\MH2_d
eth  master ≡ +1 ~1 -0 !                                   [19:54]
> python .\Alice.py
[SETUP COMPLETE]
[ALICE GUESS] = 1
[ROLL WAS]: 4
[COMMIT ACCEPTED]
[ALICE DID NOT GUESS CORRECT]
death@THATONE2  ~\Documents\Skole\ITU\5. Semester\Security 1\Assignments\Mandatory 2\Python\MH2_d
eth  master ≡ +1 ~1 -0 !                                   [19:54]
> python .\Alice.py
[SETUP COMPLETE]
[ALICE GUESS] = 5
[ROLL WAS]: 5
[COMMIT ACCEPTED]
[ALICE GUESSED CORRECT]
death@THATONE2  ~\Documents\Skole\ITU\5. Semester\Security 1\Assignments\Mandatory 2\Python\MH2_d
eth  master ≡ +1 ~1 -0 !                                   [19:54]
```