عنوان الدرس:.. اليوم:...................... التاريخ: / / ١٤

Subject:.. Day:.................. Date: / / 20 الموافق: / / ٢٠م

Q1. what the tree main properties of Zero-Knowledge proofs?

1) Completeness

2) Soundness.

3) Zero-Knowledge

Q2. Explain the roles of the prover and verifier in the Schorr protocol.

the prover: knows the secret (x) and proves knowledge of it without revealing it.

The verifier: checks the proof using math but does not learn the secret.

Q3. Why is this proof called "Zero-Knowledge"?

because the verifier knows nothing about the secret, only the prover acually knows it.

Q4. what ensure soundness in the Schorr protocol?

Soundness is ensured by the random challenge (e) acheating prover can't guess or fake valid response easily.

BLUE STAR

عنوان الدرس:........................ اليوم:........................ التاريخ: / / ١٤

Subject:........................ Day:........ Date: / /20 الموافق: / / ٢٠م

Q5 Why does the Fiat-Shamir transform remove interactivity?

because the challenge (e) is generated from a SHA hash instead of coming from the verifier

Q6. How can we detect measure the probility of successful cheating?

By running multiple cheating trials (e.g 100 times) and measure how often the cheating prover secceeds. The probabiltiy is usually is low (around 0.2)

Q7. What would happen if the challenge (e) were predictable?

cheating prover could prepare a fake valid response, breaking the security

Q8 Give one real-word use of ZKPs (e.g in blockchain privacy protocols.

ZKPs are use in cryptocurrencies like Zcash, to hide sender and receiver information while keeping transaction valid.

عنوان الدرس:........................ اليوم:........................ التاريخ: / / ١٤

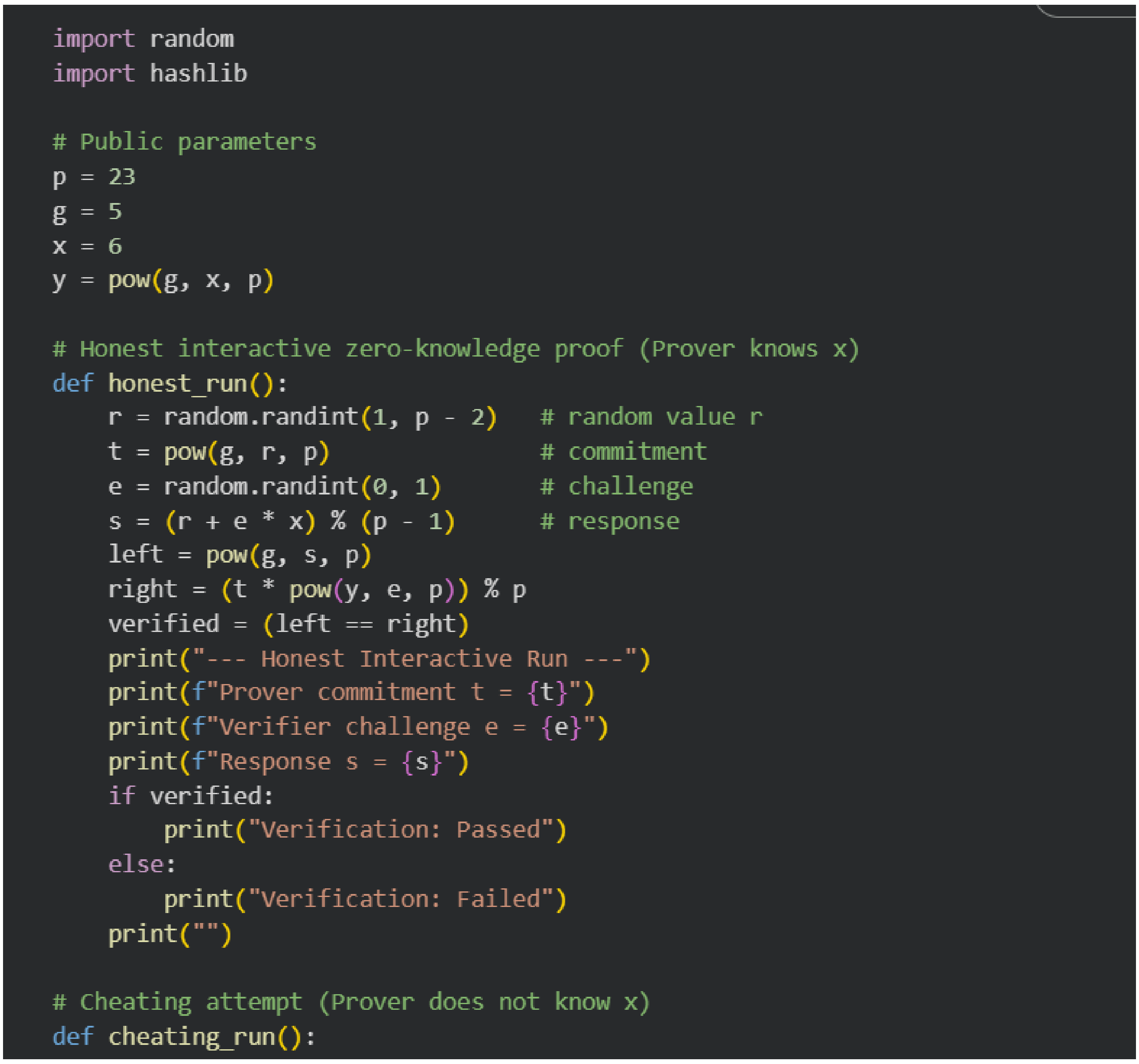
Subject:........................ Day:........ Date: / / 20 الموافق: / / ٢٠م

Q9. What is the main difference between interctive and non-interactive ZKPs?

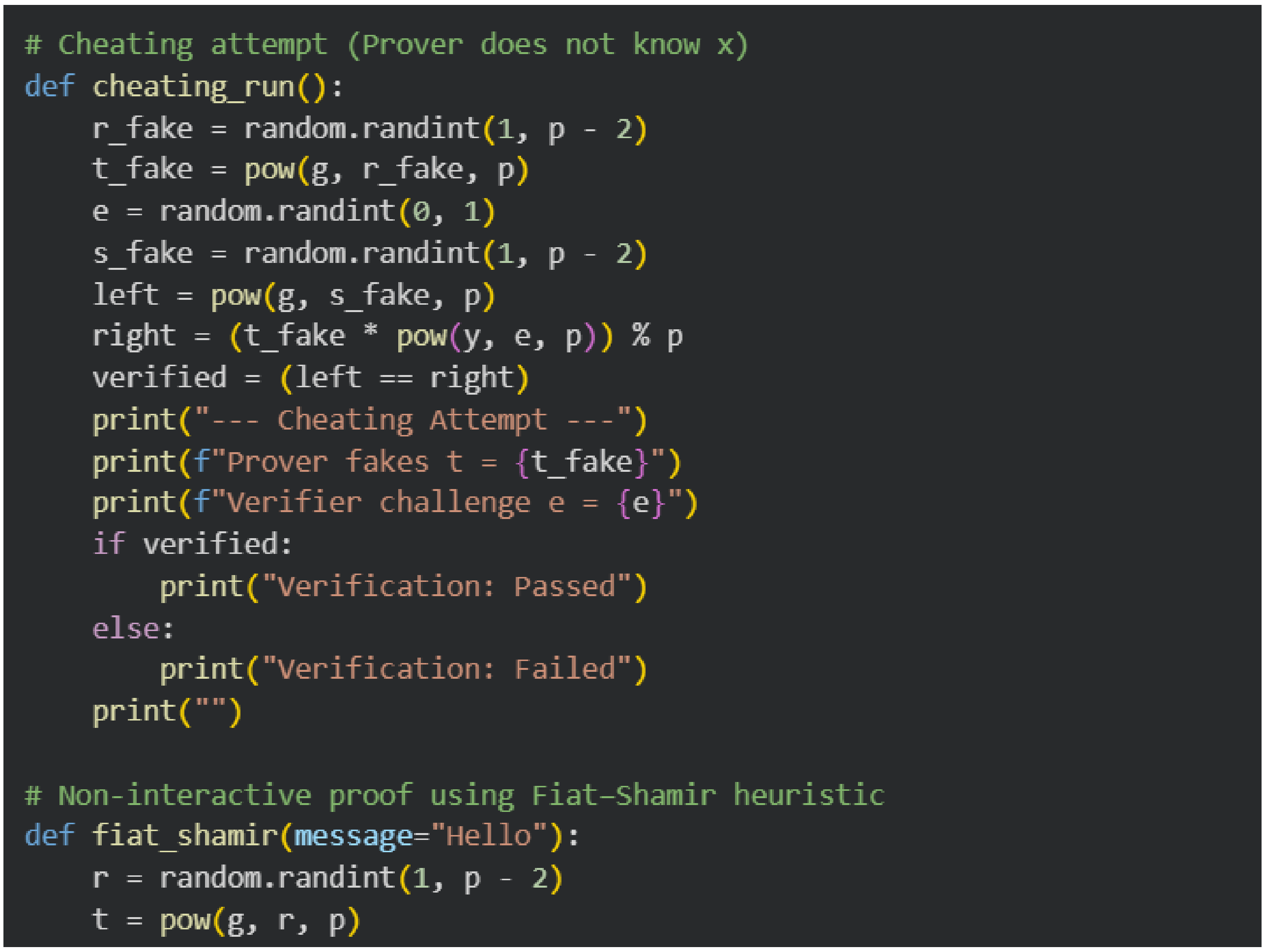
In interactive ZKPs, the prover and verifier exchange messages directly.

In non-interactive ZKPs, the challenge is generated automatically using a hash

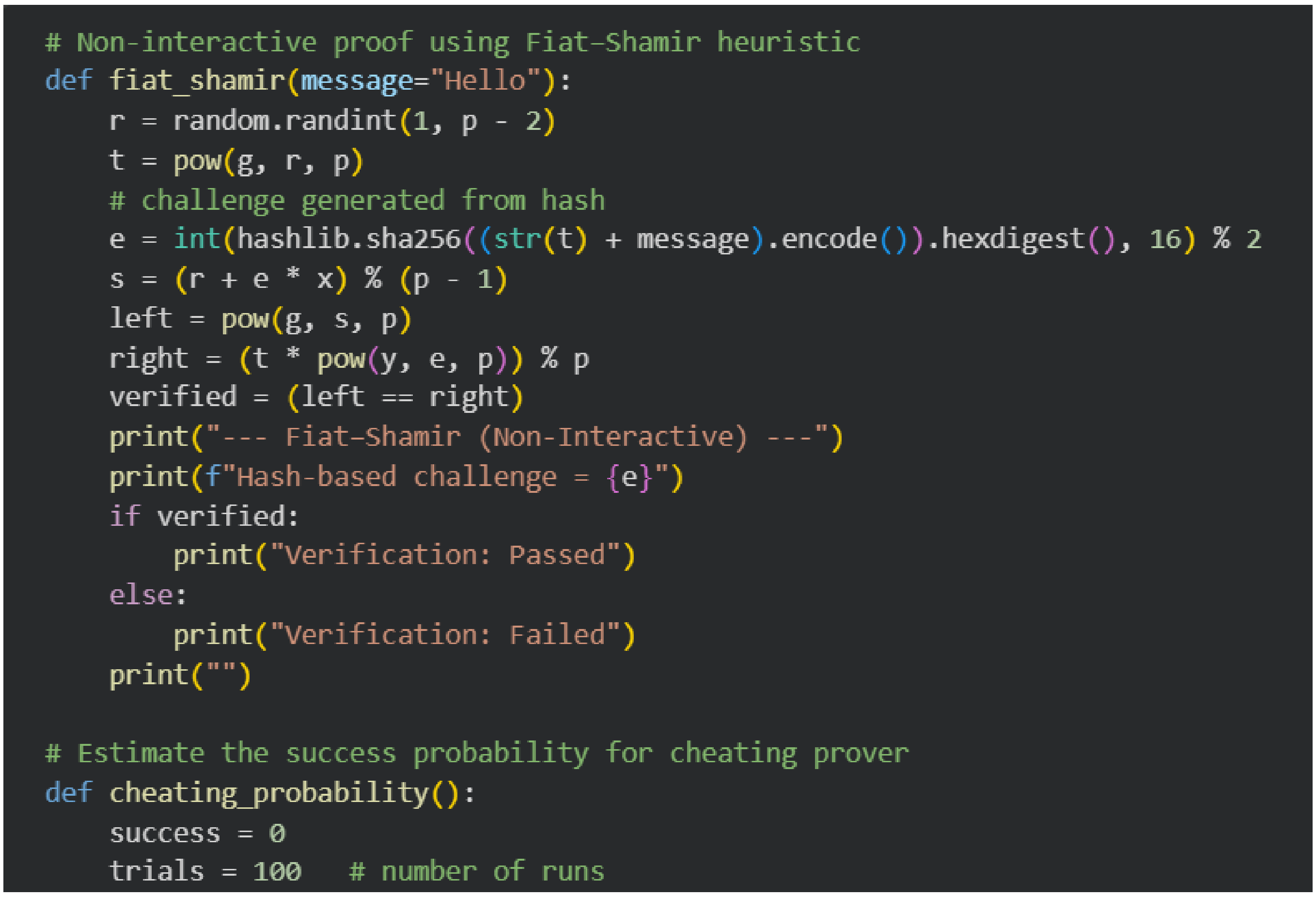
BLUE STAR

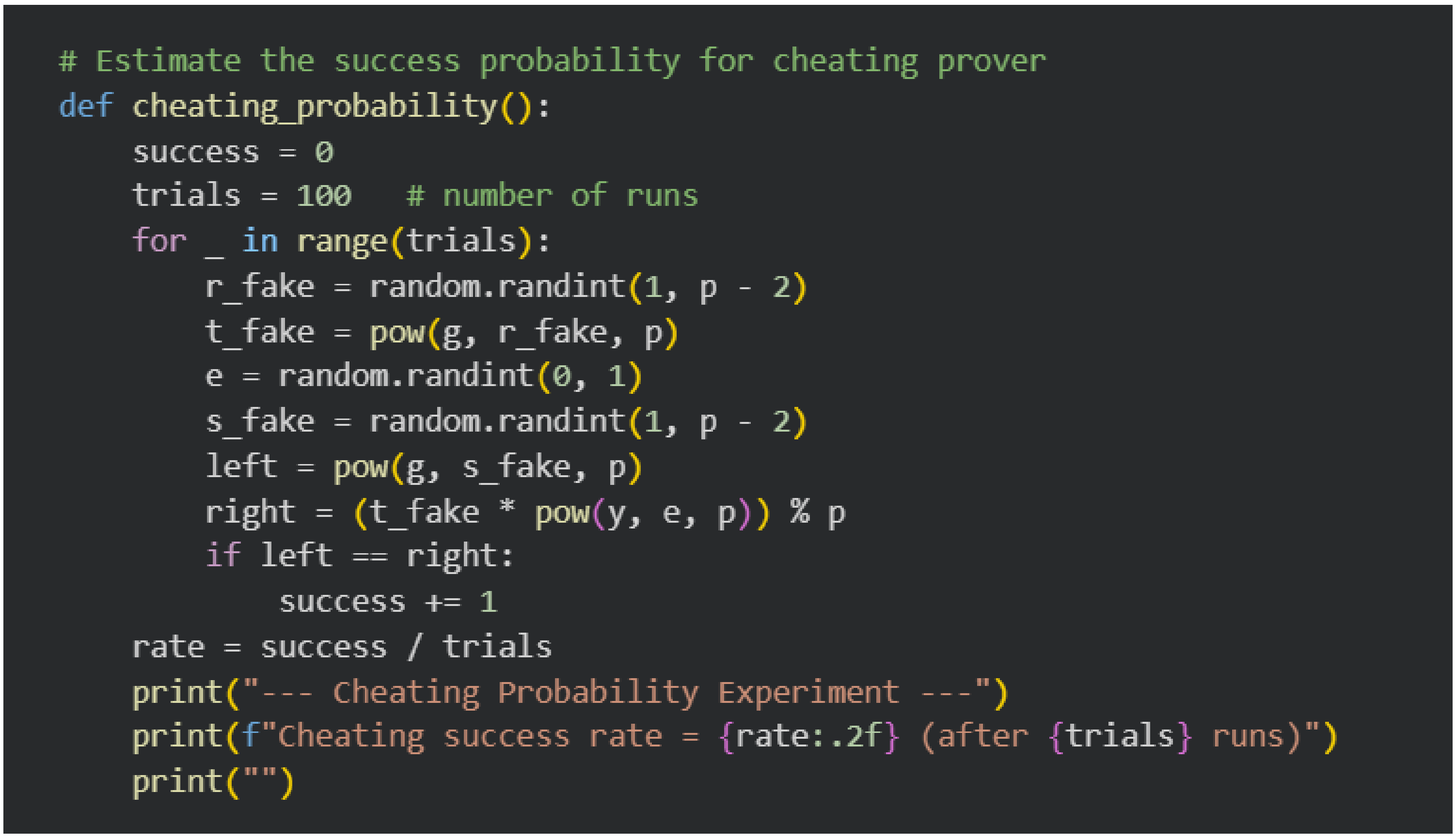


```python
import random
import hashlib

# Public parameters
p = 23
g = 5
x = 6
y = pow(g, x, p)

# Honest interactive zero-knowledge proof (Prover knows x)
def honest_run():
    r = random.randint(1, p - 2)   # random value r
    t = pow(g, r, p)               # commitment
    e = random.randint(0, 1)       # challenge
    s = (r + e * x) % (p - 1)      # response
    left = pow(g, s, p)
    right = (t * pow(y, e, p)) % p
    verified = (left == right)
    print("--- Honest Interactive Run ---")
    print(f"Prover commitment t = {t}")
    print(f"Verifier challenge e = {e}")
    print(f"Response s = {s}")
    if verified:
        print("Verification: Passed")
    else:
        print("Verification: Failed")
    print("")

# Cheating attempt (Prover does not know x)
def cheating_run():
```

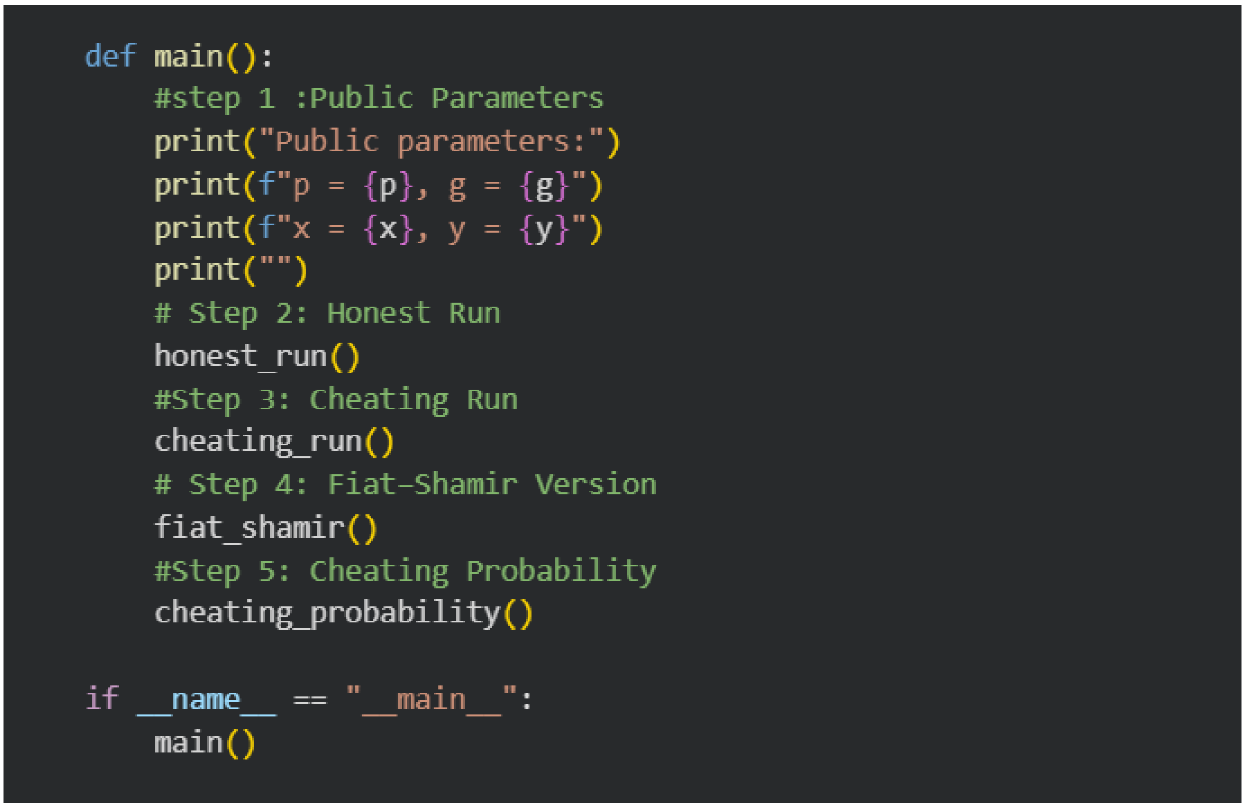


```python
# Cheating attempt (Prover does not know x)
def cheating_run():
    r_fake = random.randint(1, p - 2)
    t_fake = pow(g, r_fake, p)
    e = random.randint(0, 1)
    s_fake = random.randint(1, p - 2)
    left = pow(g, s_fake, p)
    right = (t_fake * pow(y, e, p)) % p
    verified = (left == right)
    print("--- Cheating Attempt ---")
    print(f"Prover fakes t = {t_fake}")
    print(f"Verifier challenge e = {e}")
    if verified:
        print("Verification: Passed")
    else:
        print("Verification: Failed")
    print("")

# Non-interactive proof using Fiat-Shamir heuristic
def fiat_shamir(message="Hello"):
    r = random.randint(1, p - 2)
    t = pow(g, r, p)
```

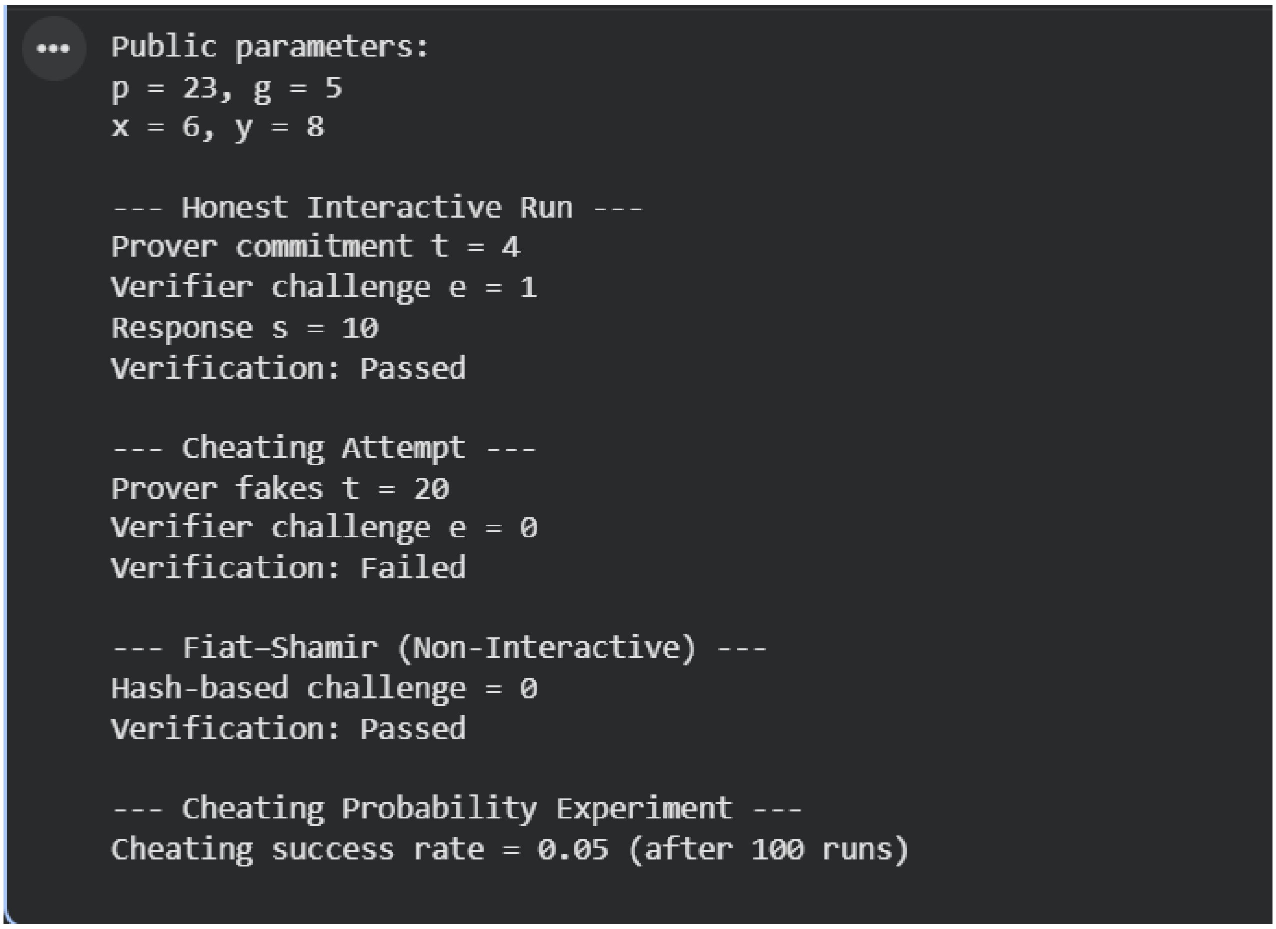

```python
# Non-interactive proof using Fiat-Shamir heuristic
def fiat_shamir(message="Hello"):
    r = random.randint(1, p - 2)
    t = pow(g, r, p)
    # challenge generated from hash
    e = int(hashlib.sha256((str(t) + message).encode()).hexdigest(), 16) % 2
    s = (r + e * x) % (p - 1)
    left = pow(g, s, p)
    right = (t * pow(y, e, p)) % p
    verified = (left == right)
    print("--- Fiat-Shamir (Non-Interactive) ---")
    print(f"Hash-based challenge = {e}")
    if verified:
        print("Verification: Passed")
    else:
        print("Verification: Failed")
    print("")

# Estimate the success probability for cheating prover
def cheating_probability():
    success = 0
    trials = 100    # number of runs
```



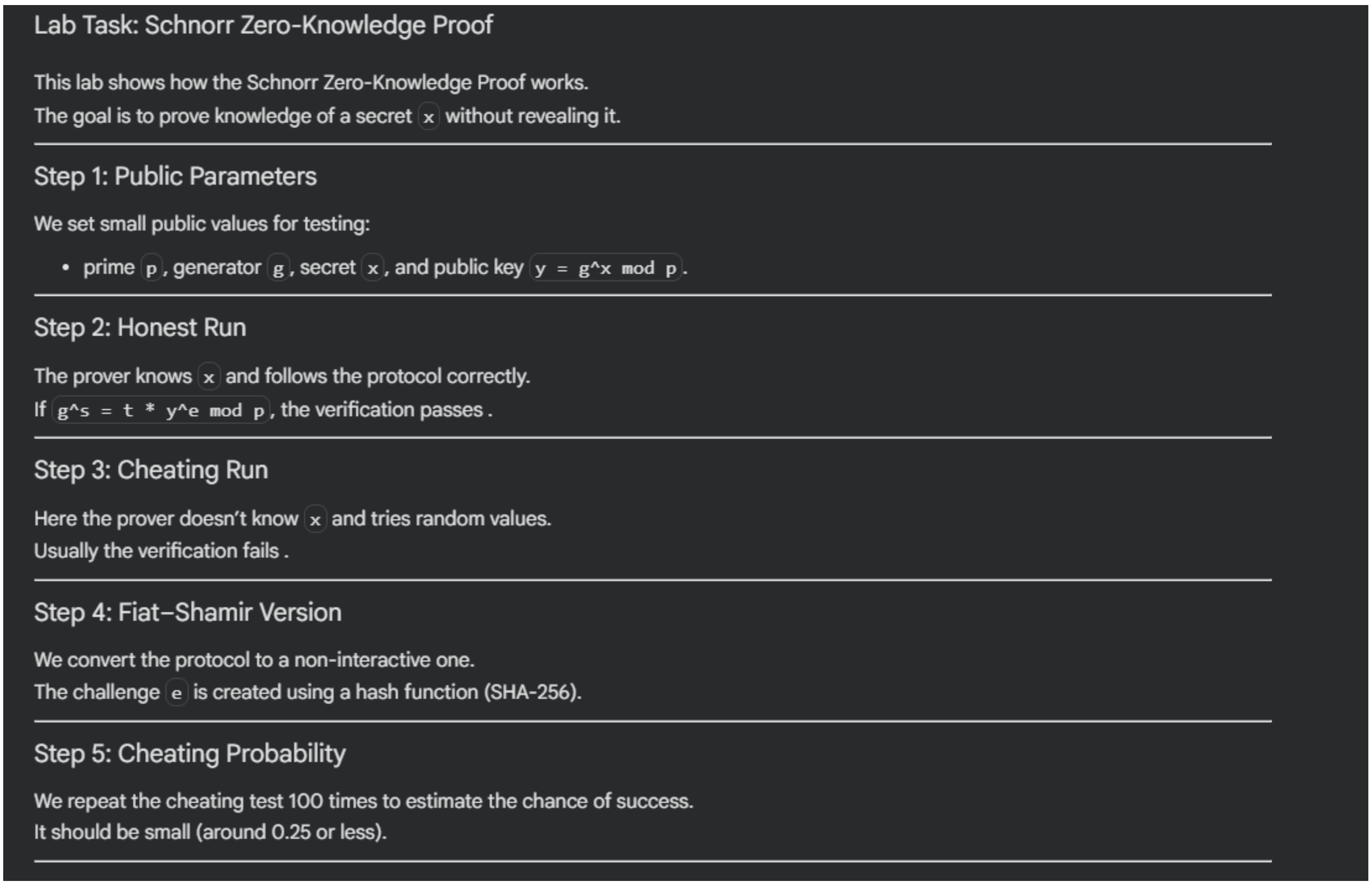
```python
# Estimate the success probability for cheating prover
def cheating_probability():
    success = 0
    trials = 100   # number of runs
    for _ in range(trials):
        r_fake = random.randint(1, p - 2)
        t_fake = pow(g, r_fake, p)
        e = random.randint(0, 1)
        s_fake = random.randint(1, p - 2)
        left = pow(g, s_fake, p)
        right = (t_fake * pow(y, e, p)) % p
        if left == right:
            success += 1
    rate = success / trials
    print("--- Cheating Probability Experiment ---")
    print(f"Cheating success rate = {rate:.2f} (after {trials} runs)")
    print("")
```



```python
def main():
    #step 1 :Public Parameters
    print("Public parameters:")
    print(f"p = {p}, g = {g}")
    print(f"x = {x}, y = {y}")
    print("")
    # Step 2: Honest Run
    honest_run()
    #Step 3: Cheating Run
    cheating_run()
    # Step 4: Fiat-Shamir Version
    fiat_shamir()
    #Step 5: Cheating Probability
    cheating_probability()

if __name__ == "__main__":
    main()
```



```
Public parameters:
p = 23, g = 5
x = 6, y = 8

--- Honest Interactive Run ---
Prover commitment t = 4
Verifier challenge e = 1
Response s = 10
Verification: Passed

--- Cheating Attempt ---
Prover fakes t = 20
Verifier challenge e = 0
Verification: Failed

--- Fiat–Shamir (Non-Interactive) ---
Hash-based challenge = 0
Verification: Passed

--- Cheating Probability Experiment ---
Cheating success rate = 0.05 (after 100 runs)
```



Lab Task: Schnorr Zero-Knowledge Proof

This lab shows how the Schnorr Zero-Knowledge Proof works.
The goal is to prove knowledge of a secret `x` without revealing it.

Step 1: Public Parameters

We set small public values for testing:

- prime `p`, generator `g`, secret `x`, and public key `y = g^x mod p`.

Step 2: Honest Run

The prover knows `x` and follows the protocol correctly.
If `g^s = t * y^e mod p`, the verification passes .

Step 3: Cheating Run

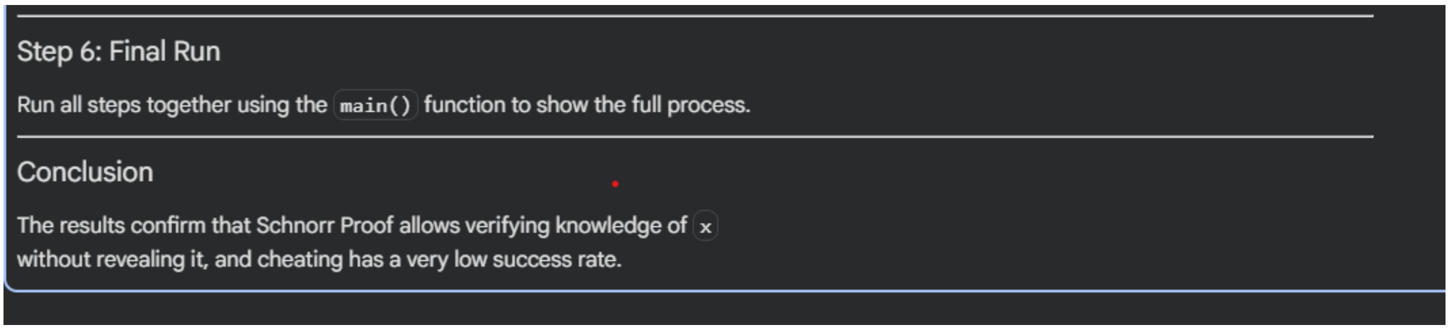
Here the prover doesn't know `x` and tries random values.
Usually the verification fails .

Step 4: Fiat–Shamir Version

We convert the protocol to a non-interactive one.
The challenge `e` is created using a hash function (SHA-256).

Step 5: Cheating Probability

We repeat the cheating test 100 times to estimate the chance of success.
It should be small (around 0.25 or less).



Step 6: Final Run

Run all steps together using the `main()` function to show the full process.

Conclusion

The results confirm that Schnorr Proof allows verifying knowledge of `x`
without revealing it, and cheating has a very low success rate.