Exercise 1

Design a client-server protocol for a file storage provider (like Dropbox). For simplicity, there is only one directory. The client should be able to execute the CRUD operations (create, read, update, delete). The server cannot be trusted, so the client should store a hash for every file in order to verify file contents fetched from the server.

The code for the server is given below. Implement the CRUD operations on the client, which will call server operations.

Which security property of the hash function does the client rely on in order not to have to trust the server?

Implementation Server

code/server.py

```
"""
Implementation Server
Server stores dictionary stored_content
key: filename
value: content
"""

stored_content = {}


def create(name, content):
    stored_content[name] = content


def read(name):
    return stored_content[name]


def update(name, content):
    stored_content[name] = content


def delete(name):
    del stored_content[name]
```

Implementation Client

code/client.py

```
"""
Implementation Client
Client stores dictionary stored_hash
key: filename
value: content-hash
"""

from hashlib import sha256
from . import server

stored_hash = {}


def create(name, content):
    stored_hash[name] = sha256(content).digest()
    server.create(name, content)


def read(name):
    content = server.read(name)
    if stored_hash[name] != sha256(content).digest():
        raise ValueError("The server lied!")
    return content


def update(name, content):
    stored_hash[name] = sha256(content).digest()
    server.update(name, content)


def delete(name):
    server.delete(name)
    del stored_hash[name]
```

Exercise 2

Write functions that produce and verify proof-of-work for SHA-256. You can use any language you like (I used Python).

There should be two functions:

1. `solve(p,d)`, which, given a puzzle string `p` and a difficulty `d` returns a puzzle solution `nonce`
2. `verify(p,nonce)`, which, given a puzzle string `p` and a puzzle solution `nonce` returns the difficulty `d` of the work that has been done to find the solution.

code/pow.py

```python
from hashlib import sha256
from sys import byteorder


def concatenate_and_hash(x, y):
    """Concatenate bytes x and int y as bitstrings,
     hash them, and return the hash as an integer. """
    y = y.to_bytes(32, byteorder)
    hash_object = sha256(x + y)
    return int.from_bytes(hash_object.digest(), byteorder)


def solve(p, d):
    """Solve a proof of work puzzle with puzzle string p of difficulty d
    and return the solution as an integer"""
    nonce = 0
    while True:
        thehash = concatenate_and_hash(p, nonce)
        if thehash < 2 ** (256 - d):
            return nonce
        nonce += 1


def verify(p, nonce):
    """Return the difficulty of a given solution to a puzzle string"""
    thehash = concatenate_and_hash(p, nonce)
    for d in reversed(range(256)):
        if thehash < 2 ** (256 - d):
            return d


puzzlestring = b'just-some-test-string'
difficulty = 20

print(f"Solving puzzle: {puzzlestring}, difficulty: {difficulty}...")
solution = solve(puzzlestring, difficulty)
print(f"Solution: {solution}")

print(f"Verifying the solution...")
verified_difficulty = verify(puzzlestring, solution)
print(f"Solution {solution} has {verified_difficulty} bits of work")
```