

```

import socket
import random

HOST, PORT = "127.0.0.1", 5004

SIZE = 8
FLEET = [4, 3, 2]
DIRS = [(0, 1), (1, 0)]

USERS_FILE = "users.txt" # simple text file: one "username:password" per line

# ----- tiny helpers (no extra libraries) -----
def load_users():
    users = {}
    try:
        with open(USERS_FILE, "r", encoding="utf-8") as f:
            for line in f:
                line = line.strip()
                if ":" in line:
                    u, p = line.split(":", 1)
                    users[u] = p
    except FileNotFoundError:
        pass
    return users

def save_user(username, password):
    with open(USERS_FILE, "a", encoding="utf-8") as f:
        f.write(f"{username}:{password}\n")

def xor_bytes(data: bytes, key: bytes) -> bytes:
    out = bytearray()
    if not key:
        return data
    for i, b in enumerate(data):
        out.append(b ^ key[i % len(key)])
    return bytes(out)

def enc_line(plaintext: str, key: bytes) -> bytes:
    ct = xor_bytes(plaintext.encode(), key)
    return ct.hex().encode() + b"\n"

def dec_line(hexline: bytes, key: bytes) -> str:
    raw = bytes.fromhex(hexline.decode().strip())
    pt = xor_bytes(raw, key)
    return pt.decode()

# ----- battleship logic -----
def new_board():
    grid = [[0]*SIZE for _ in range(SIZE)]
    ships = []

```

```

sid = 1
for length in FLEET:
    placed = False
    for _ in range(500):
        r = random.randrange(SIZE)
        c = random.randrange(SIZE)
        dr, dc = random.choice(DIRS)
        coords = []
        ok = True
        for k in range(length):
            rr, cc = r + dr*k, c + dc*k
            if not (0 <= rr < SIZE and 0 <= cc < SIZE) or grid[rr][cc] != 0:
                ok = False; break
            coords.append((rr, cc))
        if ok:
            for rr, cc in coords:
                grid[rr][cc] = sid
            ships.append({"id": sid, "size": length, "hits": 0})
            sid += 1
            placed = True
            break
        if not placed:
            raise RuntimeError("Could not place ships")
    return grid, ships

def handle_client(conn):
    f = conn.makefile("rwb")
    f.write(b"HELLO BATTLESHIP 8x8 FLEET=4,3,2\n")
    f.write(b"Commands: REGISTER user pass | LOGIN user pass | (then encrypted) NEW
| GUESS r c | STATUS | REVEAL | QUIT\n")
    f.flush()

    # --- phase 1: plaintext auth ---
    users = load_users()
    key = b""
    authenticated = False
    while not authenticated:
        line = f.readline()
        if not line: return
        parts = line.decode().strip().split()
        if not parts:
            f.write(b"ERR empty\n"); f.flush(); continue

        cmd = parts[0].upper()
        if cmd == "REGISTER" and len(parts) == 3:
            u, p = parts[1], parts[2]
            if u in users:
                f.write(b"ERR user_exists\n")
            else:
                save_user(u, p)

```

```

        users[u] = p
        f.write(b"REGISTERED\n")
    f.flush()
elif cmd == "LOGIN" and len(parts) == 3:
    u, p = parts[1], parts[2]
    if users.get(u) == p:
        # simple key from password (XOR stream key)
        key = p.encode()
        authed = True
        f.write(b"AUTH OK\n")
        f.flush()
    else:
        f.write(b"ERR bad_login\n"); f.flush()
else:
    f.write(b"ERR use REGISTER user pass OR LOGIN user pass\n"); f.flush()

# --- phase 2: encrypted game ---
grid, ships = new_board()
seen = set()
remaining = len(ships)
moves = 0

# After auth, every line is hex-encoded ciphertext
f.write(enc_line("READY encrypted_mode", key)); f.flush()

while True:
    line = f.readline()
    if not line: break
    try:
        text = dec_line(line, key)
    except Exception:
        f.write(enc_line("ERR decrypt_failed", key)); f.flush(); continue

    parts = text.strip().split()
    if not parts:
        f.write(enc_line("ERR empty", key)); f.flush(); continue

    cmd = parts[0].upper()

    if cmd == "NEW":
        grid, ships = new_board()
        seen.clear()
        remaining = len(ships)
        moves = 0
        f.write(enc_line("OK NEW", key)); f.flush()

    elif cmd == "GUESS":
        if len(parts) != 3 or not parts[1].isdigit() or not parts[2].isdigit():
            f.write(enc_line("ERR use: GUESS r c", key)); f.flush(); continue
        r = int(parts[1]); c = int(parts[2])

```

```

    if not (0 <= r < SIZE and 0 <= c < SIZE):
        f.write(enc_line("ERR out_of_bounds", key)); f.flush(); continue
    if (r, c) in seen:
        f.write(enc_line("ERR already_tried", key)); f.flush(); continue

    seen.add((r, c)); moves += 1
    cell = grid[r][c]
    if cell == 0:
        f.write(enc_line(f"MISS remaining={remaining}", key)); f.flush()
    else:
        for ship in ships:
            if ship["id"] == cell:
                ship["hits"] += 1
                sunk = (ship["hits"] == ship["size"])
                if sunk: remaining -= 1
                if remaining == 0:
                    f.write(enc_line(f"HIT SUNK size={ship['size']} WIN
moves={moves}", key))
                else:
                    if sunk:
                        f.write(enc_line(f"HIT SUNK size={ship['size']}
remaining={remaining}", key))
                    else:
                        f.write(enc_line(f"HIT remaining={remaining}", key))
                        f.flush()
                        break

        elif cmd == "STATUS":
            f.write(enc_line(f"STATUS moves={moves} remaining={remaining}", key));
f.flush()

        elif cmd == "REVEAL":
            # reveal (0/1) in multiple encrypted lines
            f.write(enc_line("BOARD", key))
            for r in range(SIZE):
                row_bits = " ".join("1" if grid[r][c] > 0 else "0" for c in
range(SIZE))
                f.write(enc_line(row_bits, key))
            f.write(enc_line("END", key))
            f.flush()

        elif cmd == "QUIT":
            f.write(enc_line("BYE", key)); f.flush()
            break

        else:
            f.write(enc_line("ERR unknown_cmd", key)); f.flush()

def main():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

```

```
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT))
s.listen(1) # one client at a time
print(f"[Server] Battleship listening on {HOST}:{PORT}")
while True:
    conn, addr = s.accept()
    try:
        handle_client(conn)
    finally:
        conn.close()

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