 Practice Final

- The exam is open book and open notes.
- You have 2 hours.
- Please answer all four questions as best you can, you may receive partial credit.
- The exam is open book and open computer. You may use course notes and documents that you have stored on a laptop, but you may NOT use the network connection on your laptop in any way, especially not to search the web or communicate with a friend.
- You are bound by the NYU honor code not to give or receive unpermitted aid.
Question 1 (32 points): Short-answer questions. Please keep your answers to a paragraph or less.

A. Why is the difficulty of the proof of work in Bitcoin set to ten minutes? What would change if it was changed to 1 minute or 1 second?

B. What is the difference between cold storage and hot storage? How might a business decide how much bitcoin to store in hot storage vs. cold storage?

C. The benefit of the Lightning Network proposal is executing payments without posting a transaction to the Bitcoin network. Can Lightning Network payments eventually replace all Bitcoin transactions completely, making the blockchain unnecessary?

D. What two conditions are required for a change to protocol rules to result in a soft fork (instead of a hard fork)?

E. Consider the following Solidity function contained in a Ethereum contract:

```
function foo(uint256 x, uint256 y) {
  while (uint256(sha3(x)) != y) {
    x++;
  }
}
```

Using an x and a y for which this function never terminates, explain how a malicious miner could use this function to waste other miners’ resources, without spending any of its own resources.

F. You’re consulting for a big-box retail store who want to implement their gift card system using blockchains. You’re considering either using colored coins or using tokens in an Ethereum contract. Name one advantage each approach has over the other.

G. Suppose your professor wants to award each student in the class a random priority for office hours help by giving them a number \( H(\text{studentID} || B_i) \) where \( B_i \) is the \( i \)th block header in the Bitcoin blockchain. Would the scheme be me more secure if you instead used \( H(\text{studentID} || B_{i-1} || B_i) \)?

H. Recall that Bitcoin uses SHA256\(^2\) as its proof of work function and that the current difficulty is set to approximately \( D = 2^{70} \). Suppose Alice discovers a way to solve the Bitcoin proof-of-work problem in expected time \( D/2^{20} \), but only for \( D < 2^{80} \). For \( D \geq 2^{80} \), the best algorithm is still brute-force, taking expected time \( D \). What is the best strategy for Alice to capitalize on this discovery?
Question 2 (22 points): (transaction signing) Recall that a Bitcoin transaction has a set of input addresses and a set of output addresses. Usually, each input address signs the entire transaction (minus the signatures) to authorize payment. This signature type is called SIGHASH_ALL.

In this question we explore other signature types where only portions of the transaction are signed. Some of these types are already supported by the Bitcoin network and some are new. Whenever a Bitcoin node validates a transaction, it checks the signatures on exactly what was signed and rejects the transaction if any of the signatures are invalid.

For each transaction signing method listed below, decide if an attacker can steal funds from an input address of a transaction submitted to the Bitcoin network. If so, explain how; if not, explain why not.

A. The secret key of each input address is used to sign the entire Txin (the input part of the transaction, minus the signatures) and nothing else. That is, the Txout (the output part of the transaction) is not signed. (this signature type is called SIGHASH_NONE)

B. The secret key of each input address is used to sign the entire Txout and nothing else. Hint: consider an address C for which there are 50 valid UTXOs that each credit C with 2 BTC (so that address C is worth 100 BTC). Is there a situation where a Bitcoin user can drain Bitcoin from address C without the owner’s authorization?

C. Suppose there are two inputs and two outputs. The secret key of the first input is used to sign the entire Txin and the first output UTXO, and nothing else. The secret key of the second input is used to sign the entire Txin and the second output UTXO, and nothing else. (this signature type is called SIGHASH_SINGLE)

D. Suppose there are two inputs and two outputs. The secret key of the first input is used to sign the first input in Txin and the first output UTXO, and nothing else. The secret key of the second input is used to sign the second input Txin and the second output UTXO, and nothing else.
**Question 3 (22 points):** (Randomized micropayments on top of Ethereum) Say, Bob runs a news site and Alice wants to micropay Bob for every article she reads. Processing all these micro-transactions on the blockchain would be inefficient. An approach to reducing the load on the block chain is called *randomized micropayments*. Here, each micropayment from Alice to Bob is worth \( x \) ether with probability \( p \), and worth zero with probability \( 1-p \). In expectation, Bob receives \( x \cdot p \) ether from each such micropayment. Because the worthless payments never hit the block chain, this enables significant transaction fee savings.

The protocol works as follows:

1. Alice sends \( 100x \) ether to an escrow contract along with her public key \( K_A \).
2. When Bob wants to request a micropayment from Alice, he sends Alice the commitment \( c = \text{SHA3}(n_B, r) \) for a random \( d \)-bit value \( n_B \) and a random 128-bit value \( r \). Bob keeps \( n_B \) and \( r \) to himself, sending only \( c \) to Alice.
3. Alice then responds with her own random \( d \)-bit value \( n_A \) and a signature \( \text{Sign}_{K_A}(n_A, c, B) \) on her nonce, Bob’s commitment, and Bob’s address \( B \).
4. Bob now checks if \( n_A = n_B \). If so, he has a winning payment and can send \( (B, n_A, n_B, c, \text{sig}_A) \) to the contract to receive \( x \) ether from the contract. If not, this is a worthless payment.
5. The contract ensures that a winning nonce \( n_B \) can only be redeemed once.

The benefit of this approach over the serial micropayment scheme we saw in the lecture, is that this single contract can be used by Alice to micropay multiple vendors.

A. How many micropayments can be processed in expectation until the contract runs out of funds? How many of these micropayments, in expectation, require writing to the block chain?
B. Explain why Alice can’t cheat Bob by choosing \( n_A \) in a way that causes Bob not to be paid. What property of SHA3 does this rely on?
C. Why is it necessary for Alice to sign Bob’s address \( B \) in step (3) of the protocol? What would go wrong if Alice’s signature did not include \( B \)?
D. Describe how Alice can execute the protocol, but then maliciously try to reclaim her own funds before Bob, whenever Bob receives a winning payment. **Hint:** in this attack, Alice simply interacts with the contract. Because of your attack, this protocol is insecure and should not be used.
**Question 5 (24 points)**: (Ethereum programming) The contract code presented below is an attempt to implement a two-player game (with a wager on the line) of Tic-Tac-Toe, also known as Noughts and Crosses:

![Tic Tac Toe board](image)

This implementation contains at least 9 bugs which compromise the security of the game. Identify 6 bugs and briefly describe how they might be fixed. Recall that Ethereum initializes all storage to zero. Assume that the function `checkGameOver()` is correctly implemented and returns true if either player has claimed three squares in a row on the current board.

1. ```
contract TicTacToe {
    // game configuration
    address[2] _playerAddress;     // address of both players
    uint32 _turnLength;        // max time for each turn

    // nonce material used to pick the first player
    bytes32 _p1Commitment;
    uint8 _p2Nonce;

    // game state
    uint8[9]   _board;             // serialized 3x3 array
    uint8 _currentPlayer;     // 0 or 1, indicating whose turn it is
    uint256 _turnDeadline;      // deadline for submitting next move

    // Create a new game, challenging a named opponent.
    // The value passed in is the stake which the opponent must match.
    // The challenger commits to its nonce used to determine first mover.
    function TicTacToe(address opponent, uint32 turnLength, bytes32 p1Commitment) {
        _playerAddress[0] = msg.sender;
        _playerAddress[1] = opponent;
        _turnLength = turnLength;
        _p1Commitment = p1Commitment;
    }

    // Join a game as the second player.
    function joinGame(uint8 p2Nonce) {
        // only the specified opponent may join
        if (msg.sender != _playerAddress[1])
            throw;
        // must match player 1's stake.
        if (msg.value < this.balance)
            throw;
        _p2Nonce = p2Nonce;
    }

    // Revealing player 1's nonce to choose who goes first.
```
function startGame(uint8 p1Nonce) {
    // must open the original commitment
    if (sha3(p1Nonce) != _p1Commitment)
        throw;

    // XOR both nonces and take the last bit to pick the first player
    _currentPlayer = (p1Nonce ^ _p2Nonce) & 0x01;

    // start the clock for the next move
    _turnDeadline = block.number + _turnLength;
}

// Submit a move
function playMove(uint8 squareToPlay) {
    // make sure correct player is submitting a move
    if (msg.sender != _playerAddress[_currentPlayer])
        throw;

    // claim this square for the current player.
    _board[squareToPlay] = _currentPlayer;

    // If the game is won, send the pot to the winner
    if (checkGameOver())
        suicide(msg.sender);

    // Flip the current player
    _currentPlayer ^= 0x1;

    // start the clock for the next move
    _turnDeadline = block.number + _turnLength;
}

// Default the game if a player takes too long to submit a move
function defaultGame() {
    if (block.number > _turnDeadline)
        suicide(msg.sender);
}
}