

---

## Final Exam

---

- The exam is open book and open notes.
- **You have 2 hours.** Please answer all five questions. All questions are weighted equally.
- 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.**
- Students are bound by the Stanford honor code.

### Question 1: Questions from all over.

- A. What is the purpose of gas in Ethereum? What would go wrong if gas were free?
- B. Why is the difficulty of the proof of work in Bitcoin set to ten minutes? What would go wrong if it was changed to 12 seconds?
- C. What is the difference between cold storage and hot storage? How would an exchange decide how much bitcoin to store in hot storage vs. cold storage?
- D. Early on, Coinbase, a large Bitcoin exchange, was facing the following type of fraud: a Coinbase user purchases bitcoin with funds from a compromised bank account and then transfers the bitcoin to an address that he or she controls. When the owner of the victim bank account complains, Coinbase has no way of getting the bitcoins back from this misbehaving user. Coinbase loses money because it has to repay the owner of the compromised bank account. How did Coinbase solve this problem?
- E. How would Bitcoin be affected if someone discovered a way to forge an ECDSA signature on an arbitrary message just given an ECDSA public key? Assume it takes 30 minutes of computation to forge a signature.
- F. When launching a new altcoin, what advantages and disadvantages would there be to using a new block chain with pre-mining for initial allocation, compared with a two-way pegged sidechain (i.e., a sidechain where assets can be transferred to and from another chain at a fixed exchange rate)?
- G. 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 block chain unnecessary?

**Question 2:** (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:** (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  $(B, n_A, n_B, c, r, \text{sig}_A)$  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 4:** (Mining pool sabotage) Recall that Bitcoin mining pools enable individual miners to share risk and reward, lowering the variance of their earnings while keeping the same expected value. Participants repeatedly submit *shares* (blocks that are valid at a lower difficulty) to prove how much work they are doing. Whenever the pool finds a block, the coinbase from that block is split among the participants in proportion to the number of shares each submitted. One risk of this is *sabotage*, in which a participant submits shares, but withholds full solutions if they are found (no coinbase is awarded for these withheld solutions).

- A. First, consider a participant with mining power  $\beta \in [0, 1]$  (as a fraction of global mining power) in a pool with total mining power  $\alpha \in [0, 1]$  (as a fraction of global mining power), where  $\beta < \alpha$ . What is the expected fraction of the overall mining rewards (the rewards collectively earned by the entire network) that this individual will earn if he or she mine honestly? Assume rewards are distributed proportionally to the number of shares submitted by each participant.  
**Hint:** there is no need for complicated expectation calculations throughout this entire question.
- B. What is the expected fraction of the overall mining rewards (the rewards collectively earned by the entire network) that this individual will earn if it devotes all of its power to sabotage?
- C. Now consider two pools,  $P_1$  and  $P_2$  with mining power  $\alpha_1$  and  $\alpha_2$ , respectively. What will  $P_1$ 's expected share of the total earnings be if it dedicates  $\beta < \alpha_1$  power towards sabotaging  $P_2$ ? Note that when  $P_1$  finds a block it gets the entire coinbase. When  $P_2$  finds a block,  $P_1$  receives a fraction of the coinbase proportional to the number of shares  $P_1$  generated while mining for  $P_2$ .  
**Hint:**  $P_2$ 's total mining power is now  $\alpha_2 + \beta$ , but only  $\alpha_2$  is used for finding a new block. Because  $\beta$  power is no longer used to find blocks,  $P_2$ 's useful mining power, as a fraction of the entire network, is now  $\alpha_2 / (1 - \beta)$ . The same reasoning also applies to  $P_1$ . You may assume that the block discovery rate is always 10 minutes.
- D. Provide concrete values for  $\alpha_1, \alpha_2, \beta$  in which this attack is profitable for  $P_1$  over honest behavior.
- E. Suppose  $P_2$  wants to protect itself by kicking out participants observed to be reporting a suspiciously low rate of valid blocks compared to how many shares they report. Explain why this might inadvertently punish honest participants.
- F. Suppose two pools, each with power  $\alpha$ , sabotage each other with power  $\beta < \alpha$ . For what range of  $\beta$  will the two pools lose revenue by attacking each other? How much will they lose?

**Note:** This is an instance of *prisoner's dilemma*: both pools are better off sabotaging if the other doesn't, but if both sabotage they end up worse off than if neither sabotaged.

**Question 5:** (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:

	X	O	X	O	X	O	X	O	X	O	X	O	X
				X		O		O		O	O	O	O
			X		X		X	X	X	X	X	X	X

This implementation contains *at least* 9 serious bugs which compromise the security of the game. Identify 4 of them 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.

```

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 the 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;
    }

    // Start the game by 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);
}
}

```